Universidad Carlos III de Madrid

Grado en Electrónica industrial y Automática

Departamento de sistemas y automática

Trabajo de Fin de Grado

Generación de algoritmos para clasificación de obstáculos

Autor: José Antonio Ramos Alonso

Tutor: Aurelio Ponz Vila

Septiembre de 2016

Agradecimientos

En primer lugar, tengo que agradecer a mi tutor Aurelio Ponz toda la ayuda y por todas las enseñanzas que me ha aportado durante el desarrollo del proyecto.

Para seguir, agradecer también a mis amigos de la universidad, en especial a Edu, Diego, Óscar y Ainoa, el haber hecho que esta etapa de mi vida sea más fácil y llevadera.

Agradecer también la inestimable ayuda de Luis e Inés, tanto cuando tenía alguna duda como por dejarme su portátil para poder realizar el proyecto.

A mis amigos del barrio, sobre todo a Javi y Kike, por ayudarme también cuando lo necesitaba. Y sobre todo a mi familia. A mis padres, por el empeño que han puesto para que estudiemos y tengamos un futuro mejor. A mis hermanas, por estar siempre ahí dando guerra. A mi tío Enrique y tía Vane. Y, sobre todo, a mi abuela María Luz, que aunque suene típico, es sin duda la mejor abuela que se puede tener.

Por todo ello y mucho más: gracias.

RESUMEN

Hoy en día, son muchas las personas que pierden la vida debido a los accidentes de tráfico. Por ello, los ingenieros de la industria automovilística buscan desarrollar nuevas tecnologías y aplicaciones que permitan reducir el número de víctimas, hasta llegar al caso ideal de que dicho número sea nulo.

Éste proyecto busca proteger a los peatones en vías urbanas e interurbanas. El objetivo es diseñar un algoritmo que, mediante la fusión de un escáner láser y una cámara en el interior del vehículo, sea capaz de detectar de manera rápida y precisa peatones en tiempo real. Además, deberá de indicar sus coordenadas al conductor, para que éste sepa dónde se encuentran dichos peatones.

Para desarrollar dicho algoritmo, se utilizan los algoritmos HOG (histograma de gradientes orientados) y SVM (máquina de soporte vectorial). El HOG es un método que se basa en la orientación del gradiente en áreas locales de una imagen para encontrar los objetos de dicha imagen. Destaca por su robustez ante los cambios de iluminación, en los fondos y en los objetos. El SVM por su parte, es un clasificador, el cual se usará para realizar el entrenamiento del HOG. Para mejorar el rendimiento, se hará un estudio de los parámetros del sistema que permiten conseguir los mejores resultados, tanto en el tiempo computacional como en la eficiencia en la detección. Por último, se recurrirá también a las librerías de OpenCV, útiles para el procesamiento de imágenes.

Palabras clave: HOG, SVM, detección de peatones, visión artificial.

Abstract

Nowadays, so many people lose their lives due to traffic accidents. For that reason, the automobile industry engineers try to develop new technologies and applications that let reduce the number of victims, until get the ideal situation which this number is zero.

This project tries to protect pedestrians in urban and intercity roads. The aim is design an algorithm that, through a fusion of a Scanner laser and a camera inside the vehicle, can be able to detect pedestrians quickly and accurately in real time. Also, it has to show the coordinates to the driver, in order to know the location of these pedestrians.

To develop this algorithm, HOG (Histogram of Orientated Gradients) and SVM (Support Vector Machine) algorithms are used. The HOG method is based on the gradient orientation in local portions of an image and is used to detect the objects of that image. It offers good results due its invariance in illumination changes, backgrounds and objects. The SVM, meanwhile, is a classifier. It will be used to train the HOG. To improve the performance, a study of the system parameters is performed in order to get the best results in computational time as well as in detection efficiency. Finally, OpenCV libraries will be used because their utility in image processing.

Keywords: HOG, SVM, pedestrian detection, artificial vision.

Índice General

Índice de figuras

Índice de tablas

SIGLAS Y ACRÓNIMOS

ABS Antilock Bracking System

ACC Adaptative Cruise Control

ADAS Advanced Driver Assistance System

DGT Dirección General de tráfico

GPS Global Positioning System

HOG Histograms of Oriented Gradients

iCab Intelligent Campus Automobile

ISA Intelligent Speed Adaptation

IVIS In-Vehicle Information System

IVVI Intelligent Vehicle based on Visual Information

KNN K-Nearest-Neighbor

Lidar Light Detection and Ranging

OpenCV Open Source Computer Vision Library

ROI Region of interest

SIFT Scale Invariant Feature Transform

SIT Sistema Inteligente de Transporte

SURF Speeded Up Robust Feature

SVM Support Vector Machine

Capítulo 1

1. Introducción y objetivos

Antes de comenzar, cabe destacar que este proyecto se ha basado en el proyecto "*Histograms of Oriented Gradients for Human Detection*" (Dalal y Triggs, 2005) [1]. Además, para la realización de la memoria se han utilizado como referencia proyectos tales como "*Desarrollo de un sistema avanzado de asistencia a la conducción en tiempo real para la detección de peatones en entornos urbanos complejos*" (Morán Natalia, 2013) [2] y "*Detección de personas*" (Intxaurbe. Jon, 2013) [3] debido a su semejanza con el nuestro. Por ello, habrá partes de estos proyectos que serán utilizadas en el desarrollo del nuestro y se irá indicando a lo largo de la memoria.

En este capítulo se pretende hacer una introducción del proyecto realizado. Se explicará el contexto en el que está ambientado así como la motivación por la que surge y los objetivos que se pretenden conseguir con dicho proyecto.

1.1 Marco Contextual

El proyecto a realizar está orientado al sector automovilístico.

El automóvil, fue creado en 1886 por Karl Benz. En sus comienzos, debido a su alto precio su utilización era escasa. Además, la máxima velocidad que podía alcanzar era de 20 Km hora. Por ambas razones, los accidentes de automóvil eran muy escasos. Sin embargo, con la llegada de la producción en cadena introducida por Henry Ford en 1910, empezó a aumentar el número de automóviles en el mercado. Por ello, su precio bajó permitiendo así que un mayor número de personas pudiesen permitirse tener un automóvil particular. Con el paso de los años y el avance de la sociedad y la tecnología se ha incrementado notablemente el número de vehículos que circulan por las carreteras, aumentando, con ello, el número de accidentes producidos.

Es debido a dichos accidentes que fallecen miles de personas todos los años. Según los datos de la Dirección General de Tráfico [4] en el último año estudiado (2014) el número de víctimas mortales fue de 1688 personas. A esto hay que sumarle también el número de personas heridas en accidentes, tanto graves como leves. En la tabla adjunta **(Tabla 1-1)** se pueden observar las cifras relativas a los accidentes de tráfico de los últimos 10 años estudiados por la DGT, así como sus consecuencias:

Año	Número de Accidentes con victimas	Total de victimas	Fallecidos	Heridos hospitalizados (graves)	Heridos no hospitalizados (leves)
2004	94.009	143.124	4.741	21.805	116.578
2005	91.187	137.251	4.442	21.859	110.950
2006	99.797	147.554	4.104	21.382	122.068
2007	100.508	146.344	3.823	19.295	123.226
2008	93.161	134.047	3.100	16.488	114.459
2009	88.251	127.680	2.714	13.923	111.043
2010	85.503	122.823	2.478	11.995	108.350
2011	83.027	117.687	2.060	11.347	104.280
2012	83.115	117.793	1.903	10.444	105.446
2013	89.519	126.400	1.680	10.086	114.634
2014	91.570	128.320	1.688	9.574	117.058

Tabla 1-1: Número accidentes de tráfico y siniestralidad durante el periodo 2004-2014 en carretera y zona urbana.

Como se puede observar en la tabla, aunque las cifras siguen siendo bastante altas, en 10 años se ha conseguido reducir considerablemente el número de fallecidos y de heridos graves en accidentes. El número de fallecidos se ha reducido en aproximadamente 3000 personas menos al año (alrededor de un 65% menos), mientras que el número de heridos de gravedad se ha reducido en cerca de 12.000 (alrededor de un 56% menos).

La mayor parte de esta reducción de victimas ha sido gracias a los avances tecnológicos que se han realizado en los vehículos. Los fabricantes de vehículos no se centran sólo en crear coches más confortables y eficientes, sino que buscan avanzar más en la reducción de accidentes, creando coches cada vez más seguros.

En un principio la seguridad de los vehículos se basaba en reducir los posibles daños en caso de accidentes (seguridad pasiva). Sin embargo con el avance de la tecnología en los últimos años se ha permitido avanzar más en temas de seguridad, consiguiendo crear sistemas que reduzcan la posibilidad de que haya un accidente (seguridad activa).

Con la intención de mejorar la seguridad global de los vehículos, se crearon los sistemas avanzados de ayuda a la conducción, llamados ADAS (Advanced Driver Assistance System) [5]. Éstos, son capaces de interactuar con el entorno y actuar sobre el mismo, lo que permite que, hoy en día, podamos hablar de coches inteligentes.

1.2 Motivación y objetivos

Como se ha visto en el apartado anterior, el número de accidentes de vehículos y sus correspondientes victimas aún es muy elevado. Es por ello que surge la motivación de crear nuevas aplicaciones y tecnologías que permitan conseguir que el número de accidentes sea cada vez menor, hasta llegar al caso ideal de que sea totalmente nulo. Esta es la principal razón por la que surge este proyecto.

El objetivo que se pretende conseguir en el proyecto presente es generar un algoritmo que nos permita la detección de peatones en vías urbanas e interurbanas. De esta forma, se podría implantar como sistema de seguridad en los vehículos con la intención de reducir el número de atropellos.

Según los datos de la DGT [4], estamos hablando de un total de 2510 peatones fallecidos en vías interurbanas y 2970 peatones fallecidos en vías urbanas en los últimos diez años estudiados por la DGT (**véanse tablas 1-2 y 1-3**).

Año	Total de victimas	Fallecidos	Heridos hospitalizados (graves)	Heridos no hospitalizados (leves)
2004	1.603	340	579	684
2005	1.551	348	575	628
2006	1.552	317	540	695
2007	1.523	287	529	707
2008	1.324	236	428	660
2009	1.247	201	368	678
2010	1.218	193	373	652
2011	1.007	158	300	549
2012	954	144	317	493
2013	945	154	278	513
2014	919	132	253	534

Tabla 1-2: Peatones víctimas en vías interurbanas

Año	Total de victimas	Fallecidos	Heridos hospitalizados (graves)	Heridos no hospitalizados (leves)
2004	10.518	343	2.136	8.039
2005	10.073	332	2.051	7.690
2006	10.214	296	1.919	7.999
2007	9.906	304	1.783	7.819

2008	9.822	266	1.634	7.922
2009	9.640	269	1.585	7.786
2010	9.705	278	1.586	7.841
2011	10.238	222	1.616	8.400
2012	10.197	232	1.599	8.366
2013	11.399	224	1.775	9.400
2014	11.944	204	1.649	10.091

Tabla 1-3: Peatones víctimas en vías urbanas

Como se puede observar en las tablas, el número de peatones victimas de accidente en vías urbanas es algo mayor que en vías interurbanas. Además, se puede apreciar que la reducción de victimas peatones en los últimos 10 años estudiados (en vías urbanas) no es tan notoria como la reducción de victimas generales mostradas anteriormente (**Tabla 1-1**). Mientras que la reducción de víctimas mortales a nivel general fue de un 65% en los últimos 10 años, la reducción de víctimas mortales peatones es de un 41% en vías urbanas.

Esto se debe a que, a pesar de todas las medidas actuales que hay de seguridad en vehículos, la gran parte está orientada a proteger a los usuarios de dentro del vehículo, dejando con menor protección a los usuarios externos, tales como motoristas, ciclistas y peatones. Por esta razón, surgen proyectos como este.

Nuestro algoritmo se encargará de detectar peatones en el entorno de un vehículo que circula por vías urbanas e interurbanas. Se dispondrá para ello de un láser y una cámara en el interior del vehículo. También indicará las coordenadas reales de los peatones respecto al vehículo de tal forma que el conductor sepa en todo momento la localización de dichos peatones.

Por último, al estar trabajando en tiempo real y tratarse de la vida de personas, se quiere conseguir que el tiempo de computación sea el menor posible para conseguir cuanto antes los resultados, de tal forma que el conductor pueda actuar lo antes posible para evitar un posible accidente.

El algoritmo ha sido desarrollado sobre el sistema operativo Linux/Ubuntu a través de una máquina virtual llamada VirtualBox. Dicho algoritmo ha sido realizado en los lenguajes de programación C/C++, basándose en las librerías OpenCV [6] en el entorno de desarrollo integrado Qt Creator.

Capítulo 2

2. Estado del arte

El Estado del Arte del presente documento se centra en explicar al lector el estado actual de la seguridad en los automóviles, explicando para ello algunos de los sistemas de seguridad existentes. Además, se hará referencia a los vehículos capaces de moverse con completa autonomía y se explicará también las plataformas de las que se dispone en la Universidad Carlos III para proyectos como este.

2.1 Seguridad en los automóviles

2.1.1 Seguridad activa y pasiva

Debido a la gran cantidad de accidentes que ocurren en las carreteras, cada vez son más los fabricantes de vehículos que tratan de aumentar la seguridad de los vehículos a diseñar, con la intención de evitar dichos accidentes o proteger a los usuarios de dichos accidentes. Por ello, actualmente existen muchos sistemas de seguridad que permiten reducir el número de accidentes o que, en su defecto, minimizan los daños producidos si dicho accidente no se puede evitar. El problema de estos sistemas, es que la mayoría se centran en proteger a los usuarios de dentro del vehículo. Así, personas como los motoristas, peatones o ciclistas se quedan sin protección ante un accidente con un vehículo. Es por ello que en la actualidad se busca desarrollar sistemas que permitan proteger también a los usuarios externos.

En temas de seguridad en vehículos, podemos hablar de seguridad activa y seguridad pasiva.

- **SEGURIDAD ACTIVA**

Es el conjunto de todos aquellos elementos que contribuyen a proporcionar una mayor eficacia y estabilidad al vehículo en marcha, y en la medida de lo posible, evitar un accidente [7]. Forman parte de la seguridad activa elementos tales como: sistemas de frenado (siendo el más común el ABS), sistema de dirección, sistema de suspensión, amortiguadores, neumáticos, sistemas de control de estabilidad e iluminación.

- **SEGURIDAD PASIVA**

Es el conjunto de todos aquellos elementos que reducen los daños en caso de que el accidente sea inevitable [7]. No evita el accidente, pero reduce las consecuencias. Forman parte de la seguridad pasiva elementos tales como: cinturones de seguridad, airbags, reposacabezas, además del chasis y la carrocería capaces de absorber la mayor parte de la energía en caso de impacto.

2.1.2 Sistema ADAS

Los sistemas ADAS (Advanced Driver Assistance System) son Sistemas Avanzados de Asistencia al Conductor. Estos sistemas son desarrollados para aumentar la seguridad global en los vehículos y proporcionar una conducción más confortable.

Los sistemas ADAS, divididos en sistemas de ayuda al conductor y sistemas de ayuda al vehículo, son capaces de advertir de problemas de seguridad al conductor y a los ocupantes del vehículo e incluso tienen capacidad de actuación y reacción automática en situaciones de riesgo elevado donde el conductor no haya reaccionado a tiempo, de manera que permiten al vehículo tomar decisiones por sí mismo [8]. Estos sistemas pueden prevenir hasta un 40**%** de accidentes [9], dependiendo del tipo de sistema y de la situación.

Dependiendo de su funcionalidad, podemos distinguir los siguientes tipos de sistemas ADAS:

- Sistemas destinados a dar soporte a varios aspectos de la conducción proporcionando información, conocidos como IVIS (In-Vehicle Information System).
- Sistemas de alerta al conductor, encargados de disminuir los fallos humanos.
- Sistemas que intervienen en el control del vehículo (sin llegar a suplantar al conductor).

A continuación se describen una serie de sistemas ADAS, de tal forma que se pueda apreciar mejor la variedad de estos y su utilidad.

- **SISTEMAS DE VISION NOCTURNA**

Los sistemas de visión nocturna son aquellos dispositivos que en ausencia de luz nos permite ver más allá de lo que ilumina el propio alumbrado del coche [10]. Dependiendo del fabricante existen diversos matices en su funcionamiento aunque el dispositivo opera de forma similar. Una cámara, de infrarrojos o termográfica, situada en el frontal del vehículo, envía una señal que es proyectada en el tablero de instrumentos o en la pantalla del navegador **(véase Figura 2-1).** De esta forma, el conductor del vehículo puede ver con mayor claridad si existe alguna persona, animal u obstáculo en la vía. Consiguiendo así disminuir el riesgo de accidente.

Figura 2-1: Sistema de visión nocturna [10]

- **SISTEMAS DE ASISTENCIA AL APARCAMIENTO**

Los sistemas de ayuda al aparcamiento llevan tiempo en el mercado, sin embargo, cada vez se van incorporando a más modelos. Los sistemas de asistencia al aparcamiento son aquellos que facilitan la maniobra de aparcamiento al conductor [11].

Para evitar al conductor la tensión de maniobrar en espacios muy ajustados, se usan sensores de aparcamiento. Estos sensores utilizan una tecnología de ondas ultrasónicas que detecta la distancia respecto de otros coches u objetos (**véase Figura 2-2**). Están situados discretamente en los paragolpes delantero y trasero, y avisan al conductor mediante un pitido y mediante alertas visuales en un monitor.

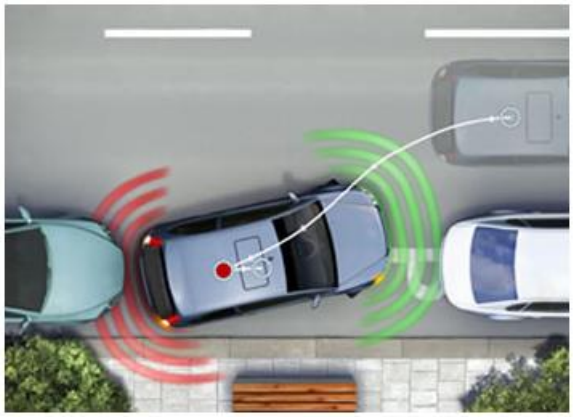


Figura 2-2: Sistema de asistencia al aparcamiento [11]

De esta forma, se consigue que el conductor sea capaz de estacionar su vehículo en huecos más estrechos y de evitar posibles choques con los otros vehículos.

Además, en la actualidad existen coches (como por ejemplo algunos de la marca Toyota) que llevan incorporados como sistema adicional una cámara de visión trasera (**véase Figura 2-3**) [12].

Figura 2-3: Cámara de visión trasera [12]

Al poner la marcha atrás, aparece en el salpicadero una gran imagen a color de la parte trasera del coche que muestra claramente la proximidad respecto a cualquier obstáculo. Para facilitar aún más el aparcamiento, unas líneas de guiado vinculadas al volante muestran el recorrido previsto hasta el hueco de aparcamiento de tal forma que el conductor sepa exactamente cuánto ha de girar el volante.

- **SISTEMA ADAPTATIVO DE LUCES**

Los sistemas adaptativos de luces son aquellos que permiten una iluminación automática de la carretera sin la intervención del conductor y sin perjuicio de los demás usuarios de la vía, todo ello gracias a una cámara que detecta si tenemos vehículos que vienen de frente o que nos preceden en la calzada [13].

Esto permite detectar posibles obstáculos antes de que entren en nuestra zona de visión. El sistema inteligente de iluminación reconoce también el tráfico que circula en sentido contrario y simplemente disminuye la intensidad en esa zona **(véase Figura 2-4).**

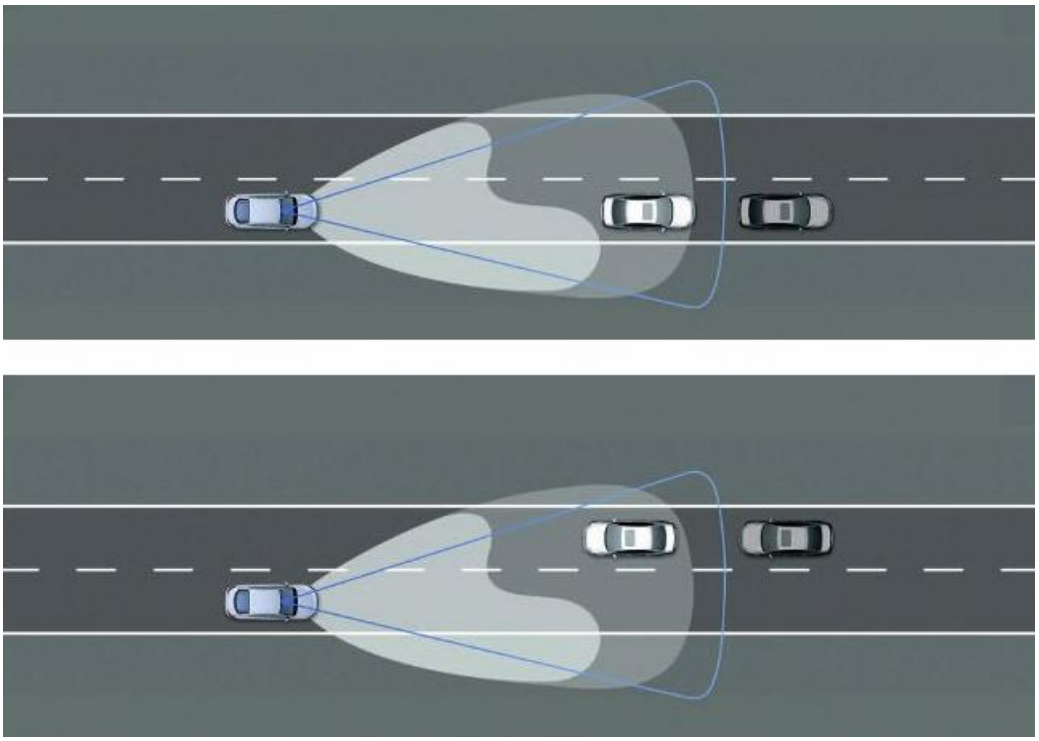


Figura 2-4: Ejemplo de funcionamiento del sistema adaptativo de luces [13]

Además, algunos de estos sistemas incorporan sistemas de infrarrojos para aumentar la seguridad, pudiendo detectar obstáculos y otros usuarios de la vía a una distancia mucho mayor.

- **SISTEMA DE ADAPTACIÓN INTELIGENTE DE VELOCIDAD (ISA)**

Un gran número de accidentes de tráfico son debido a que los vehículos circulan por encima de los límites de velocidad establecidos, ignorando las señales de tráfico de la vía por la que circulan. Controlando la velocidad del vehículo se podría reducir en gran medida el número de estos accidentes y la gravedad de los que se producen.

Los Sistemas de Adaptación inteligente de velocidad (conocidos por las siglas ISA, Intelligent Speed Adaptation) son sistemas que "conocen" los límites de velocidad permitidos o recomendados en cada tramo de la calzada, poniendo esta información a disposición del conductor o incluso llegando a limitar la velocidad del vehículo a la máxima permitida [14].

A diferencia del sistema de control de velocidad, que incorpora ya la mayoría de los coches y que permite al conductor establecer una velocidad máxima a la que desea circular, el ISA ayuda al conductor a mantener la velocidad dentro de los límites recomendados. A través de un software que analiza las imágenes de una cámara y reconoce las señales de tráfico y combinando tecnología GPS de información del tráfico en tiempo real, el ISA determina los límites de velocidad establecidos y los recomendados en cada momento, avisando al conductor mediante señal acústica y/o visual en el caso de que dicho límite sea sobrepasado [15]. El sistema podría incluso actuar directamente sobre el vehículo, ajustando automáticamente su velocidad.

- **CONTROL DE VELOCIDAD DE CRUCERO ADAPTATIVO (ACC)**

El control de crucero adaptativo es una evolución del control de crucero [16]. Los sistemas de control de crucero permiten mantener una velocidad constante previamente fijada por el conductor que se desconectará automáticamente si este pisa el freno.

El ACC por su parte, se muestra más inteligente y tiene también en cuenta el tráfico a la hora de mantener la velocidad, con lo que el conductor no tendrá que estar acoplándose continuamente a las condiciones de la carretera.

El control de crucero adaptativo cuenta con una serie de radares que se encargan de detectar el tráfico en la vía, de tal manera que si nos encontramos con un coche por delante a una velocidad inferior, automáticamente el sistema alerta al conductor del peligro y reduce la velocidad de nuestro vehículo actuando sobre el sistema de frenos, de forma que se mantiene la distancia de seguridad que haya sido predeterminada. Una vez que el carril por el que circulamos queda libre, el sistema acelera el vehículo hasta la velocidad que hayamos programado.

- **SISTEMA DE DETECCIÓN DE ÁNGULO MUERTO**

Uno de los riesgos más comunes en la conducción de un vehículo es la falta de visibilidad en los retrovisores. Existe un ángulo o un punto muerto en el que el conductor no puede percibir a otro vehículo durante la marcha, poniendo en riesgo a los demás usuarios.

En el 2007, Volvo desarrolló un sistema de seguridad activa BLIS (Blind Sport Information System) [17], cuyo significado es sistema de información sobre el ángulo muerto. Este sistema detecta si un vehículo entra en nuestro ángulo muerto, alertándonos de su presencia.

Con el paso de los años, el resto de marcas como Mercedes-Benz, Ford, Opel, Citroën, Volkswagen entre otras han ido incorporando el sistema en sus modelos.

El sistema consta de una cámara en cada uno de los retrovisores del vehículo que registra 30 fotografías por segundo. Las fotografías son comparadas entre sí, detectando de este modo la intrusión de un vehículo, ya sea moto o coche, en el ángulo muerto del vehículo.

Además dispone de un sistema de aviso que consta de unos diodos luminosos que se encuentran en la parte interior de la puerta a la altura del retrovisor, tal como indica en la siguiente imagen (**Figura 2-5**):

Figura 2-5: Sistema de aviso de ángulo muerto [17]

El área de detección del sistema es el siguiente (**Figura 2-6**):

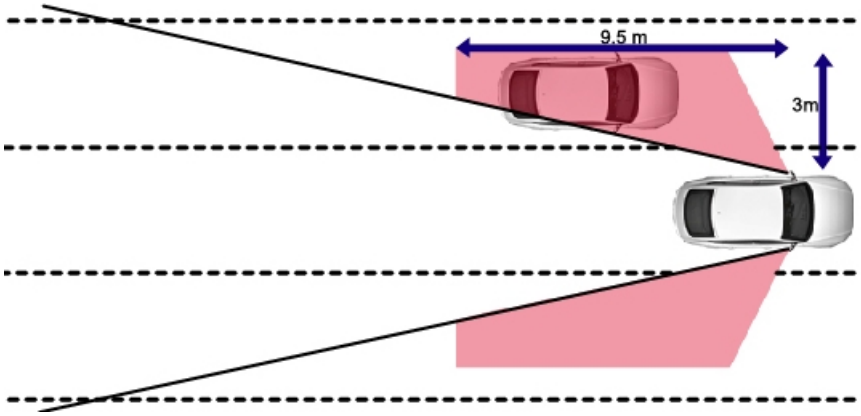


Figura 2-6: Área de detección del ángulo muerto [17]

Cuando un vehículo se encuentre dentro del área de detección, el sistema avisa al conductor por medio de los diodos.

- **SISTEMA ABS**

El Sistema ABS (Anti-lock braking system) es un dispositivo utilizado en aviones, automóviles y en modelos avanzados de motocicletas que hace variar la fuerza de frenado para evitar que los neumáticos pierdan la adherencia con el suelo [18].

En el sector de la automoción es uno de los mayores avances en seguridad activa. El ABS permite al conductor mantener el control de la trayectoria del vehículo en caso de que se produzca una frenada brusca, con la consiguiente posibilidad de poder esquivar posibles obstáculos mediante el giro del volante de dirección (**véase figura 2-7)**.

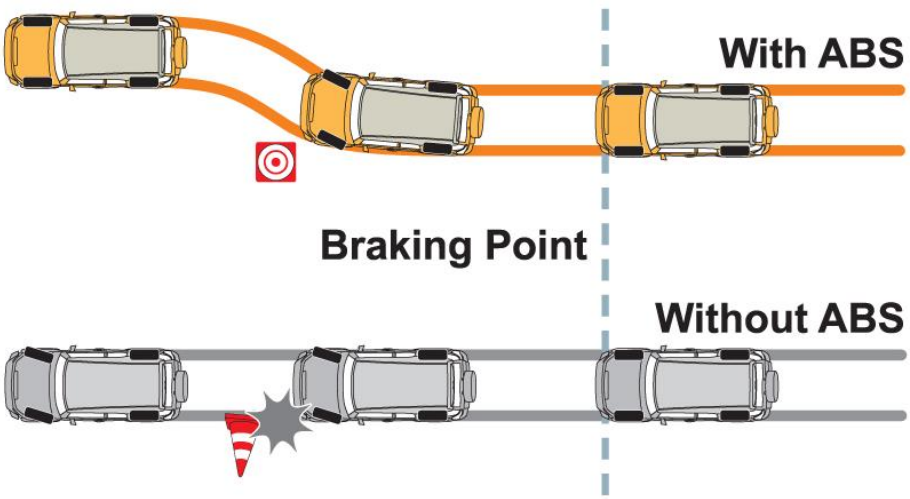


Figura 2-7: Movimiento de vehículo tras frenado con ABS y sin ABS [19].

El ABS funciona en conjunto con el sistema de frenado tradicional. Consiste en una bomba que se incorpora a los circuitos del líquido de freno y en unos detectores que controlan las revoluciones de las ruedas. Si en una frenada brusca una o varias ruedas reducen repentinamente sus revoluciones, el ABS lo detecta e interpreta que las ruedas están a punto de quedar bloqueadas sin que el vehículo se haya detenido. Esto quiere decir que el vehículo comenzará a deslizarse sobre el suelo sin control, sin reaccionar a los movimientos del volante. Para que esto no ocurra, los sensores envían una señal al Módulo de Control del sistema ABS, el cual reduce la presión realizada sobre los frenos, sin que intervenga en ello el conductor. Cuando la situación se ha normalizado y las ruedas giran de nuevo correctamente, el sistema permite que la

presión sobre los frenos vuelva a actuar con toda la intensidad. El ABS controla nuevamente el giro de las ruedas y actúa otra vez si éstas están a punto de bloquearse por la fuerza del freno. En el caso de que este sistema intervenga, el procedimiento se repite de forma muy rápida, unas 50 a 100 veces por segundo, lo que se traduce en que el conductor percibe una vibración en el pedal del freno.

- **SISTEMAS DE PROTECCION PARA PEATONES**

Como se ha podido observar en los puntos anteriores, la gran parte de sistemas de seguridad que se utilizan en la actualidad están orientados a proteger a los usuarios de dentro del vehículo y permitirles una conducción más segura y confortable. Sin embargo, existen pocas tecnologías que permitan proteger a los usuarios externos, tales como los ciclistas, motoristas y peatones. En caso de accidente, estos últimos son los que salen peor parados. Estamos hablando de que en Europa el porcentaje de peatones que muere por accidente de coche es de un 15 % [20]. Por ello, aunque son escasos, existen algunos sistemas de seguridad para evitar el atropello de peatones o reducir las consecuencias si el accidente es inevitable.

En el caso de que el accidente sea inevitable, se pretende conseguir que el vehículo absorba la mayor parte del impacto para que el peatón sufra el menor daño posible. Por ello, los fabricantes desarrollan soluciones tales como carrocerías deformables, que permitan reducir de forma drástica la gravedad de las lesiones en el atropellado. Además, se han creado también sistemas de actuadores que elevan el capo del vehículo en caso de atropello. Mediante sensores y algoritmos inteligentes, se detecta la colisión y en el caso de que sea un peatón se eleva el capó tal y como se muestra en la **Figura 2-8** [21].

Figura 2-8: Capó elevable para protección de peatones [21]

También se puede destacar como sistema de reducción de daños los airbags externos al vehículo, que funcionan de forma similar a los internos, con la diferencia de que estarían orientados a proteger a los usuarios externos al vehículo.

Todos estos sistemas, como ya se ha comentado, están destinados a reducir los daños en caso de atropello. Sin embargo, ninguno de ellos evita que se produzca el accidente. Es por ello que los estudios actuales se centran en desarrollar sistemas que eviten al vehículo chocar con el peatón o reducir el golpe disminuyendo su velocidad.

Dichos sistemas se basan en utilizar una cámara en el interior del vehículo y algoritmos de procesamiento de procesamiento de imágenes de tal forma que dicho vehículo sea capaz de detectar a los peatones que están en su trayectoria y avisar al conductor para evitar el atropello. Además, en caso de que el conductor no sea capaz de evitar el accidente, el sistema tendría también la posibilidad de realizar una frenada de emergencia automática. Teniendo en cuenta que la protección de los peatones es el tema a tratar en este proyecto, se explicará más detalladamente este sistema en los siguientes capítulos.

2.2 Vehículos Autónomos

Tal y como se ha visto en el apartado anterior, muchos fabricantes desarrollan sistemas ADAS que permiten tomar el control al vehículo (sin llegar a suplantar al conductor) para evitar accidentes.

Sin embargo, algunos fabricantes han ido más allá desarrollando un vehículo de conducción completamente automática, siendo el más conocido, el vehículo autónomo de Google.

VEHÍCULO AUTÓNOMO DE GOOGLE

El *Google Driverless Car* [22] es un proyecto realizado por Google que se basa en el desarrollo de las tecnologías necesarias para conseguir coches que circulen de forma autónoma, sin la presencia de un conductor. El líder del proyecto es el ingeniero alemán Sebastian Thrun, director del Stanford Artificial Intelligence Laboratory y coinventor de Google Street View.

Figura 2-9: Vehículo autónomo de Google [23]

Actualmente, este coche (**Figura 2-9**) es capaz de conducir por sí solo, reconocer carriles, señales de tráfico, vehículos, ciclistas y peatones, controla la distancia de seguridad con el vehículo que va delante y toma las decisiones pertinentes para no tener ningún percance.

Para llevar a cabo todas estas acciones, el coche lleva incorporado un conjunto de diversas tecnologías. Entre ellas, destacan las siguientes:

- Un sensor de orientación: con él, rastrea el movimiento y equilibrio del coche.

- Un radar, encargado de medir la velocidad de los vehículos que circulan por delante.
- Un procesador, el cual lee datos y regula el comportamiento del coche.
- Un sensor láser, el cual escanea 360º alrededor del vehículo.
- Un sensor de rueda, utilizado para detectar el número de rotaciones y ayudar a determinar la ubicación del coche.

2.3 Plataformas de investigación de la Universidad Carlos III de Madrid

Con el fin de probar los algoritmos que se desarrollan en el Laboratorio de Sistemas Inteligentes, la Universidad Carlos III de Madrid dispone de dos vehículos conocidos como IVVI e iCab, cuyas características y componentes fueron recopilados en [2].

2.3.1 Vehículo IVVI

El vehículo IVVI (Vehículo Inteligente basado en Información Visual, en sus siglas en inglés), es un coche real que incorpora sistemas de ayuda a la conducción basados en el análisis computacional de las imágenes.

Los sistemas que incorpora son los siguientes:

- **Sistema estéreo en blanco y negro de barrido progresivo**: permite la captura de imágenes en movimiento, evitando los problemas inherentes al vídeo entrelazado.
- **Cámara de infrarrojo lejano**: encargada de captar el calor que desprenden obstáculos tales como peatones o vehículos.
- **Cámara a color**: utilizada para percibir información tal como la ofrecida por las señales de tráfico.
- **Iluminación infrarroja:** permite trabajar de noche sin molestar al resto de ocupantes del vehículo.
- **Cámara enfocada al conductor**: se utiliza para evaluar el grado de atención del mismo.
- **GPS**: encargado de suministrar información al resto de sistemas de percepción sobre la orientación y velocidad del vehículo.
- **Dos ordenadores** situados en el maletero del vehículo para el procesamiento de la información captada por los sistemas de visión por computador.

- **Convertidor DC/AC** conectado directamente a la batería del vehículo para obtener la alimentación eléctrica necesaria para el funcionamiento de los equipos y sensores.

Figura 2-10: Vehículo IVVI 2.0 [24]

El IVVI 2.0 (**Figura 2-10**) es el segundo vehículo adquirido por la universidad con el equipamiento necesario para desarrollar Sistemas de Ayuda a la Conducción. Se diferencia del primer vehículo en que además de emplear equipos más modernos, presenta los ordenadores, sensores y monitores integrados en el vehículo. Entre dichos equipos, destacan un láser para detección de obstáculos y generación de ROIs y una cámara para clasificación. Ambos serán los utilizados en este proyecto y por tanto, se explicarán en los siguientes capítulos.

2.3.2 Icab

El iCab (cuyo significado es Intelligent Campus Automobile) es un proyecto diseñado para la implementación de un Sistema Inteligente de Transporte (SIT). Se basa en la modificación de un coche eléctrico de la marca EZGO (**Figura 2-11**). En él, se han cambiado la tracción y la dirección, con tal de conseguir un control automático, usando para ello un ordenador o, en su defecto, de forma manual mediante unos pedales y un joystick. Gracias al uso de un GPS, un telémetro láser y sistemas de visión infrarroja y estereoscópica, este coche tiene la capacidad de conocer los entornos estructurados e interactuar con ellos.

Figura 2-11: Vehículo iCab [25]

Capítulo 3

3. Fundamentos teóricos

En este capítulo se explicaran los fundamentos teóricos a los que se ha recurrido durante el desarrollo del proyecto. En ellos, se podrán ver las diferentes alternativas que se podrían utilizar para el desarrollo de un proyecto como éste. En los siguientes capítulos, se explicarán en mayor profundidad los utilizados en dicho proyecto.

3.1 Tecnología para la obtención de información

Como hemos podido comprobar en los apartados anteriores, los sistemas ADAS pueden basarse en el uso de tecnologías tanto laser, radar, y otros como en visión por computador. Además, existen sistemas que pueden usar ambos para conseguir una mejor información del medio.

Los láser, por su parte (en concreto los láser lidar [26]) suelen utilizarse en los sistemas de asistencia a la conducción gracias a su resolución y su alta precisión. Sin embargo, la cantidad de información que se obtiene es menor que la que conseguiríamos con un sistema de visión por computador. Este último capta la información en 3D mientras que los láser la captan solo en 2D, lo que supone una desventaja a la hora de clasificar los objetos que se han detectado. Por ello, se suele recurrir a la unión de ambos.

En nuestro caso, se recurre un proyecto realizado por los profesores del departamento llamado "*Automatic Obstacle Classification Using Laser Scanner and Camera Fusion*" [27] implementado en el IVVI 2.0. Este proyecto se basa en la combinación de un escáner láser y una cámara para la detección de obstáculos y la clasificación de éstos. Por un lado, el escáner láser permite la detección de obstáculos en condiciones de baja iluminación, cuando la cámara no puede ofrecer ayuda. Ésta última, por su parte, permite la clasificación de los objetos detectados gracias a su capacidad 3D.

La combinación de ambos nos permite generar las ROIs (regiones de interés) de una imagen para que luego éstas últimas puedan ser usadas para clasificación de obstáculos. Esta aplicación se ha utilizado en nuestro proyecto, por ello se explicará más detalladamente en el capítulo 4.

3.2 Técnicas para la detección y clasificación de objetos

En este apartado se hace una recopilación de las diferentes técnicas que podemos utilizar para llevar a cabo la detección y clasificación de objetos. Siendo estos, en nuestro caso, peatones.

Cabe destacar que el desarrollo de dicha recopilación ha sido facilitado gracias a proyectos tales como [2] y [3].

3.2.1 Técnicas para la extracción de características

Estas técnicas se basan en la extracción de características de una imagen para usarlas en el proceso de clasificación de datos. Para corregir posibles fallos en los datos (debido a posibles errores en los sensores) se suele llevar a cabo un preprocesado antes de la extracción de dichas características. Además, dicho preprocesado puede usarse también para preparar los datos para posteriores etapas de extracción o clasificación.

Las características que son elementales, es decir, aquellas que están presentes explícitamente en los datos obtenidos, pueden pasar a la etapa de clasificación de manera directa. Sin embargo, las características de un orden alto derivan de dichas características elementales y son obtenidas gracias a la manipulación o transformación en los datos. A continuación, explicamos de manera breve los más destacados:

- **DESCRIPTORES WAVELET DE HAAR**

Los descriptores Wavelet de Haar son un método propuesto en 2004 por Viola y Jones [28]. Destacan por su robustez a la hora de definir objetos complejos, ya que son invariantes a cambios en los colores o en las texturas. Por ello, son utilizados de manera habitual para llevar a cabo la detección de personas.

El método de los descriptores Wavelet de Haar consiste en aplicar varios clasificadores a una ventana de detección, los cuales se usan en serie y son, cada cual, más complejo que el anterior. Esta última ventana, se encarga de recorrer una imagen completa y, cuando encuentre un objeto, utilizará dichos clasificadores para comprobar sus características y confirmar si dicho objeto es el que se desea encontrar. En el caso de que el objeto no cumpla ninguna de las condiciones de los clasificadores, el proceso no continúa. Sin embargo, si dicho objeto pasa todos los filtros significa que es el objeto buscado.

Cabe destacar que el algoritmo se puede ejecutar en varias escalas, permitiéndonos así encontrar objetos de diferentes tamaños o, incluso, de tamaños desconocidos.

• SIFT (SCALE-INVARIANT FEATURE TRANSFORM)

Este descriptor fue descrito por David G. Lowe, de la universidad de la Columbia Británica en Vancouver [29]. Es un método utilizado para obtener características de imágenes que sean invariantes tanto a escala como rotación y parcialmente inmune a la iluminación como al punto de vista de la cámara. Además dichas características se obtienen tanto en el dominio espacial como en el frecuencial, reduciendo así la probabilidad de disrupción por oclusión o ruido.

• SURF (SPEEDED UP ROBUST FEATURE)

El algoritmo SURF (Speeded-Up Robust Features) [30] es uno de los sucesores más importantes de SIFT. SURF fue presentado en 2006 en el ECCV en Graz (Austria).

Está basado en los mismos principios y pasos que el SIFT, pero utiliza un esquema diferente que ha demostrado conseguir mejores resultados.

El SURF detecta los puntos de interés a partir del cálculo del DoH (determinante de la matriz Hessiana) y los describe mediante el Wavelet de haar. Con ello se consigue una respuesta mejor (en cuanto a la descripción de las imágenes) de la que se conseguiría usando SIFT, además de más rápida, gracias al uso del DoH en lugar del DoG (la usada por el SIFT).

• DETECTORES DE BORDES CANNY

El Algoritmo de Canny es un operador desarrollado por John F. Canny en 1986 que utiliza un algoritmo de múltiples etapas para detectar una amplia gama de bordes en imágenes [31]. Esta técnica, se caracteriza por estar optimizada para la detección de bordes diferenciales y consta de tres etapas principales: filtrado, decisión inicial, e histéresis.

• HOG

El método HOG (Histogram of Oriented Gradients) es un método presentado por Navneet Dalal y Bill Triggs en 2005 [1]. Se basa en la orientación de los gradientes de una imagen para la detección de objetos existentes en dicha imagen. Es un método similar al SIFT [29], Reconocimiento de formas (Belongie 2001) [32] e Histograma de bordes orientados, (Freeman and Roth 1995, [33]), entre otros. La diferencia que

presenta HOG es que éste no calcula los gradientes de manera uniforme sobre un mallado denso, sino que primero divide la imagen en bloques, estos a su vez en sub-bloques y una vez obtenido estos últimos, se calcula en ellos el gradiente y el histograma. Una vez sacado el histograma de gradientes orientados, se almacenan en un vector de características. Este procedimiento supone una mejora en los resultados en comparación con los métodos anteriores. Por esta razón, ha sido el que se ha elegido para el desarrollo de nuestro proyecto y será explicado en profundidad en el capítulo 4.

3.2.2 Clasificadores

Los clasificadores son algoritmos que tienen la capacidad de aprender la distribución de datos de un conjunto de ejemplos de entrenamiento para posteriormente predecir la clase a la que pertenecen nuevas muestras que no se han utilizado en dicho entrenamiento. En función del aprendizaje, se pueden distinguir clasificadores de aprendizaje supervisado y de aprendizaje no supervisado.

3.2.2.1 Aprendizaje Supervisado

El aprendizaje supervisado es una técnica de aprendizaje artificial utilizada para deducir una función matemática a partir de datos de entrenamiento [34].

Los datos de dicho entrenamiento están formados por pares de objetos (normalmente vectores), una componente de ese par son los datos de entrada y, el otro, los resultados deseados. La salida de la función puede ser un valor numérico (como en los problemas de regresión) o una etiqueta de clase (como en los de clasificación).

El objetivo del aprendizaje supervisado es el de crear una función capaz de predecir el valor correspondiente a cualquier objeto de entrada después de haber visto una serie de ejemplos (los datos de entrenamiento). Es decir, a partir de un conjunto de ejemplos se genera una función que relaciona las entradas con las salidas de interés. Para que dicha función sea precisa, se deben de utilizar una gran cantidad de ejemplos de entrenamiento.

Entre las técnicas de aprendizaje supervisado, destacan las siguientes:

• ADAPTIVE BOOSTING

Las técnicas de Boosting se basan en combinar clasificadores individuales, de tal forma que se consigan mejores resultados que con dichos clasificadores por separado. Clasificadores iniciales, que por separado pueden considerarse clasificadores débiles, se utilizan en serie uno tras otro en forma de cascada, actuando cada uno sobre las predicciones del anterior, consiguiendo que el clasificador resultante tenga resultados fuertes (Paul Viola y Michael Jones) [35].

• ALGORITMO KNN

El algoritmo KNN (K-Nearest-Neighbor) [36], también llamado algoritmo del vecino más cercano, es un método que sirve para clasificar información y obtener predicciones. Sirve para estimar la probabilidad de que un elemento x pertenezca a la clase C.

El KNN toma la información que se desea clasificar, calcula la distancias de todos los miembros de la base de datos (es decir, del conjunto de entrenamiento) y coge una cantidad K de vecinos, los cuales son los que se encuentran a una distancia menor. Posteriormente, estos K vecinos son los usados para la clasificación.

• SUPPORT VECTOR MACHINES (SVM)

Las máquinas de soporte vectorial, máquinas de vectores de soporte o máquinas de vector soporte (Support Vector Machines, SVMs) son un conjunto de algoritmos de aprendizaje supervisado desarrollados por Vladimir Vapnik y su equipo en los laboratorios AT&T [37]. Dado un conjunto de ejemplos de entrenamiento (de muestras) podemos etiquetar las clases y entrenar una SVM para construir un modelo que prediga la clase de una nueva muestra

Tomando los datos de entrada como conjuntos de vectores en un espacio n-dimensional, una máquina de vectores soporte construirá un hiperplano de separación en ese espacio. Se considera que es mejor clasificador de datos aquel hiperplano que maximice la distancia (o margen) con los puntos que estén más cerca de él. Siendo los vectores de soporte los puntos que tocan el límite del margen. En el contexto que se está tratando en este proyecto de detección de personas, las clases de datos corresponderán al objeto (muestras positivas), mientras que el resto de la imagen será tachada como muestras negativas. Al igual que el HOG, el SVM será explicado más en detalle en el siguiente capítulo, pues es el otro algoritmo utilizado en el presente proyecto.

3.2.2.2 Aprendizaje No supervisado

En este caso, a diferencia del aprendizaje supervisado, el conjunto de entrenamiento utilizado no dispone de etiquetas. Por ello, es necesario utilizar técnicas de agrupamiento que se encarguen de crear dichas etiquetas. Entre estas técnicas, destacan Hidden Markov Models (HMM) [38], K-means [39] o self-organizing maps (SOM) [40].

Capítulo 4

4 Desarrollo del proyecto

El presente capítulo se centrar en explicar el desarrollo del proyecto realizado. Como ya se ha comentado anteriormente, el proyecto está basado en la utilización de los algoritmos HOG y SVM. Por ello, para una mejor comprensión del lector, en la primera parte de este capítulo se explicarán los conceptos teóricos y matemáticos de dichos algoritmos. Tras esto, se explicará la aplicación de dichos algoritmos en el proyecto. Además, se indicará cómo se han identificado las regiones de interés de las imágenes y cómo se ha desarrollado la lógica de un etiquetador para la comprobación de las coordenadas indicadas por el algoritmo.

4.1 Bases teóricas

4.1.1 Descriptores HOG- Histograma de gradientes orientados

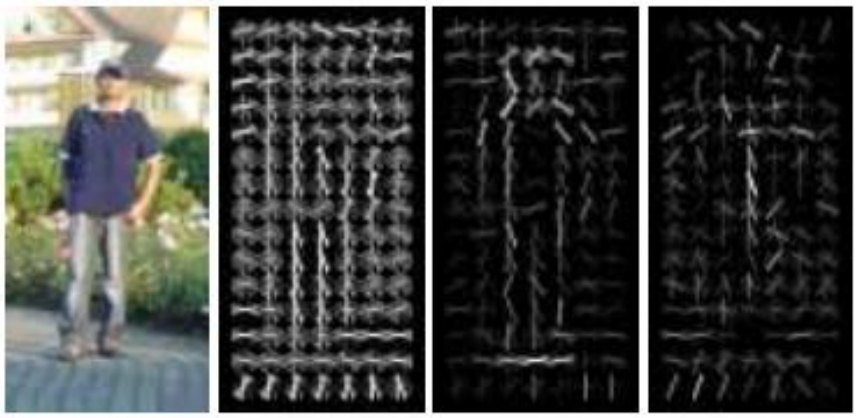
Los descriptores HOG (del inglés Histogram of Oriented Gradients) se basan en la orientación del gradiente en áreas locales de una imagen. El descriptor HOG permite aprovechar de forma eficiente la información del gradiente (**Figura 4-1b**) a partir de combinar esta información en forma de histogramas orientados locales, que se calculan en celdas de pequeño tamaño, las cuales se distribuyen de forma uniforme por toda la imagen (**véase Figura 4-1c**). Dichos histogramas nos proporcionan información de las orientaciones de los contornos que dominan en cada una de las posiciones de la imagen. Esta información nos va a permitir distinguir la forma de los objetos presentes en una imagen y es una buena base para detectar y reconocer dichos objetos. Gracias a esa información, podemos ver la frontera entre un objeto y otro.

(a) (b) (c)

Figura 4-1: Gradientes e histogramas de una imagen: a) Imagen original, b) Gradientes de la imagen, c) Histograma de Gradientes Orientados [41]

Una vez obtenido los histogramas, para conseguir mejores resultados, los histogramas calculados en cada celda se agrupan en bloques de un tamaño un poco mayor. Estos bloques sirven para normalizar la representación final y hacerla más invariante a los cambios de iluminación y a distorsiones en la imagen. Así, la representación final será la concatenación de la representación de todos estos bloques. La razón de hacer la normalización en bloques en lugar de hacerla en la imagen completa se debe a que puede ocurrir que los cambios de iluminación no sean uniformes en la imagen, habiendo zonas de la imagen con mayor o menor iluminación que las otras.

Estos últimos bloques que han sido normalizados, son lo que los autores llaman descriptores HOG (**Figura 4-2).**



(a) (b) (c) (d)

Figura 4-2: Ejemplo de extracción de descriptores HOG: a) Imagen original; b) Descriptores HOG, c) Descriptores HOG en positivo; d) Descriptores HOG en negativo [1]

Estos descriptores nos proporcionan información tal como los cambios de intensidad debido a los contornos o bordes de una imagen, y se utilizarán como entrada a un clasificador SVM. Para calcular dichos descriptores, se siguen los siguientes pasos (**véase figura 4-3**):

1º Teniendo una imagen a color, se pasa a escala de grises

2º Tras esto, se calculan los gradientes en la imagen completa.

3º Una vez calculados se divide la imagen en bloques, solapados cierta área. Para llevar a cabo el avance de bloques, se elimina la columna de las celdas de la izquierda y se van añadiendo en la columna de la derecha. Por otro lado, en el eje vertical, se eliminan las filas de las celdas superiores y se añaden filas inferiores.

4º Se dividen los bloques en sub-bloques o celdas.

5º Se calcula en cada uno de ellos el histograma de gradientes orientados

6º Se aplica una ventana gaussiana sobre cada bloque, y se almacena la información en un vector de características.

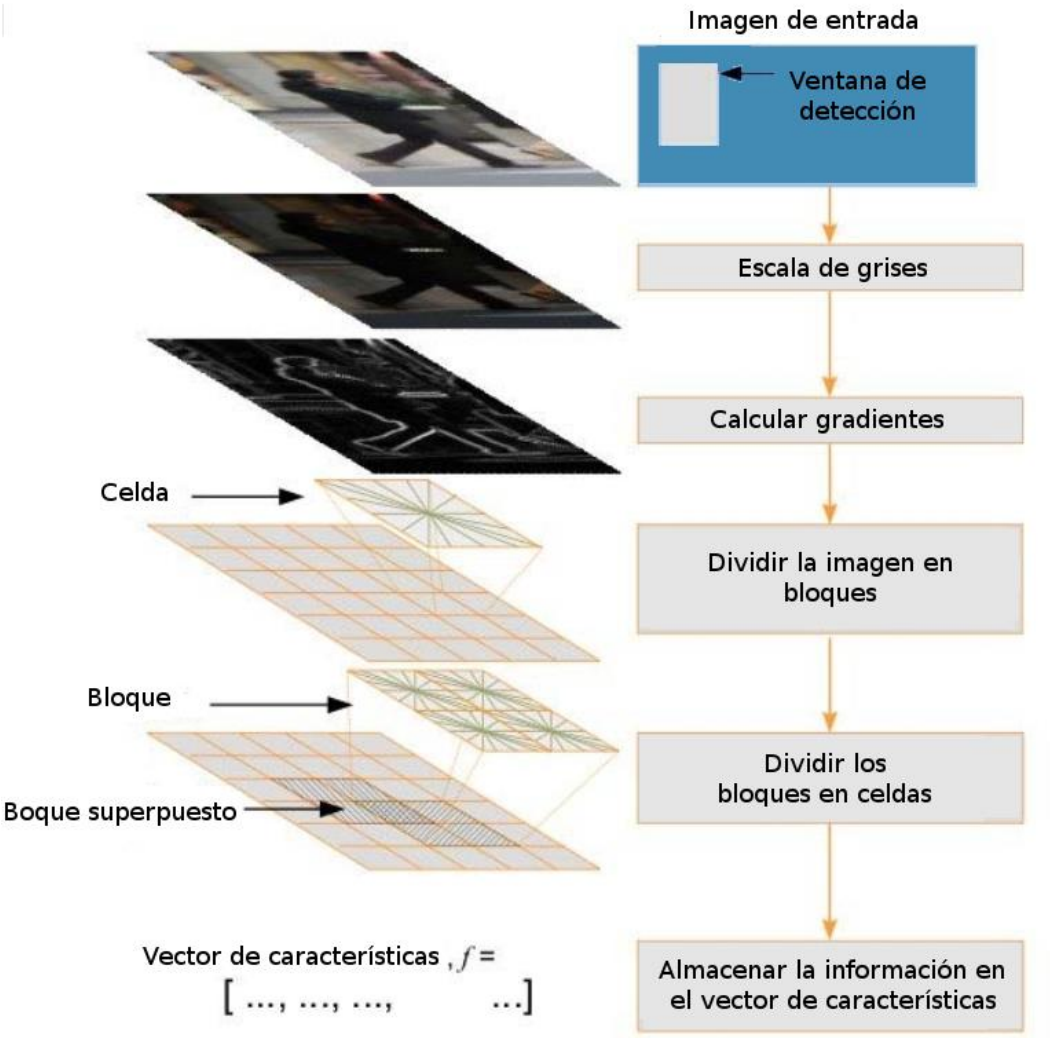


Figura 4-3: Proceso de extracción de características de una imagen [3]

Estos vectores de características son los llamados descriptores HOG. Nos permiten obtener información de una imagen y ver las diferentes fronteras que separan un objeto de otro, haciendo su detección más sencilla.

Por todo esto, el descriptor HOG es bastante adecuado para llevar a cabo la detección de peatones, ya que nos permitirá poder diferenciar dichos peatones en una escena por la distinta orientación de sus gradientes con los del fondo. Además, como se ha comentado ya, destaca por su robustez ante las distintas condiciones de iluminación, cambios en el contorno y diferencia de fondos y objetos.

Además, , cabe mencionar (tal y como se refleja en [3]) que los fundamentos teóricos de los métodos HOG se basan en proyectos tales como descriptores SIFT [29], Histogramas de bordes orientados [33], y reconocimiento de formas [32]. Sin embargo, estos métodos no son iguales. En los tres primeros se calculan los gradientes de forma

uniforme sobre un mallado denso. En el caso del HOG, sin embargo, primero se divide la imagen en bloques y sub-bloques. Tras esto, se calcula en ellos los gradientes y el histograma, mejorando con ello, el rendimiento [1].

La idea a seguir para nuestro proyecto será detectar a los peatones mediante el HOG, el cual deberá ser entrenado antes por el SVM (dicho entrenamiento se explicará en los siguientes apartados). Tras esto, se desea que nuestro algoritmo indique las coordenadas de dichos peatones al conductor del vehículo. Este aviso se pretende que sea lo antes posible, puesto que se está tratando de la vida de personas. Cuanto antes se obtenga la información más tiempo tendrá el conductor para tomar las decisiones oportunas para evitar el accidente. Por ello, debido a que el cálculo del HOG supone un alto coste de memoria, en este proyecto se intentará reducir el coste computacional de tal forma que la detección se realice de la manera más rápida posible.

DESCRIPCIÓN MATEMÁTICA

Para entender mejor el funcionamiento del HOG, procedemos a explicar su base matemática.

El HOG se basa en la orientación de los gradientes de una imagen, por ello, explicaremos primero la parte matemática correspondiente al cálculo de los gradientes.

El gradiente se define como un cambio de la intensidad de la imagen en una cierta dirección [39]. Dicha dirección es aquella en la que el cambio de intensidad es máximo. Se calcula para cada uno de los pixeles de una imagen y queda definido para cada pixel por dos valores:

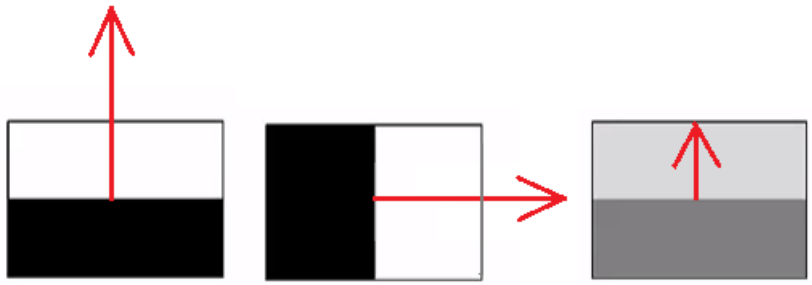
- La dirección donde el cambio de intensidad es máximo
- La magnitud del cambio en dicha dirección.

Por ejemplo, se tienen las tres siguientes imágenes (**Figura 4-4**):

(a) (b) (c)

Figura 4-4: Muestra de imágenes con distintos cambios de intensidad [41]

Cogemos en cada una de ellas el pixel central, y procedemos a dibujar su gradiente (flecha roja):



(a) (b) (c)

Figura 4-5: Representación de gradientes [41]

Como se puede observar, en la primera imagen (**Figura 4-5a**) se tiene un gradiente en dirección vertical, pues es la dirección donde el cambio de intensidad es máximo (pasamos de negro a blanco). En la tercera sin embargo (**Figura 4-5c**), se produce en la misma dirección pero su magnitud es menor, puesto que no hay tanto cambio de intensidad como en la primera (pasamos de un gris a otro con diferente valor). En la segunda imagen (**Figura 4-5b**) por su parte, se tiene un gradiente de la misma magnitud que el primero, pero con dirección horizontal, puesto que es la dirección donde hay mayor cambio de intensidad en dicha imagen.

El cálculo de estos gradientes se realiza a partir de la diferencia de intensidad de los pixeles vecinos en dirección tanto horizontal como vertical. Por ejemplo, si tenemos la imagen A (**Figura 4-6**):

0	0	255	255	255
0	0	255	255	255
0	0	0	255	255
0	0	0	0	0
0	0	0	0	0

Figura 4-6: Imagen A [41]

Y queremos calcular el gradiente en el punto central, debemos de aplicar la siguiente relación:

$$dx = A(x+1,y) - A(x-1, y) \quad \textbf{(4.1)}$$

$$dy = A(x, y+1) - A(x, y-1) \quad \textbf{(4.2)}$$

Dónde:

- dx: diferencia horizontal entre el pixel anterior y el siguiente al seleccionado.

- dy: diferencia vertical entre el pixel anterior y el siguiente al seleccionado.

A partir de estas diferencias se puede calcular la orientación y magnitud global del gradiente.

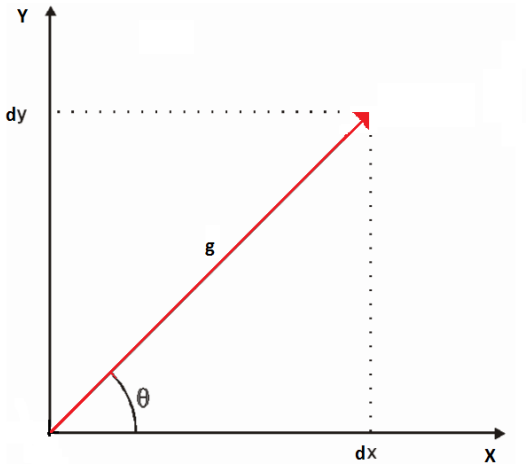


Figura 4-7: Vector Gradiente [41]

Este vector **(Figura 4-7**) es que el que se corresponde con el vector gradiente para este pixel. A partir de este vector podemos calcular la orientación del gradiente en un punto y la magnitud de dicho gradiente en ese punto.

$$\text{Orientación} \rightarrow \Theta(x,y) = arctan\frac{dy}{dx} \quad \textbf{(4.3)}$$

$$\text{Magnitud} \rightarrow g(x,y) = \sqrt{dx^2 + dy^2} \quad \textbf{(4.4)}$$

Una vez que tenemos el gradiente, explicamos cómo se calcularía el HOG.

Para calcular el HOG se siguen dos pasos fundamentales:

1º Se divide la imagen en un número fijo de celdas y para cada una de esas celdas se obtiene un histograma de las orientaciones de los gradientes en esa celda [42] (véase Figura 4-8).

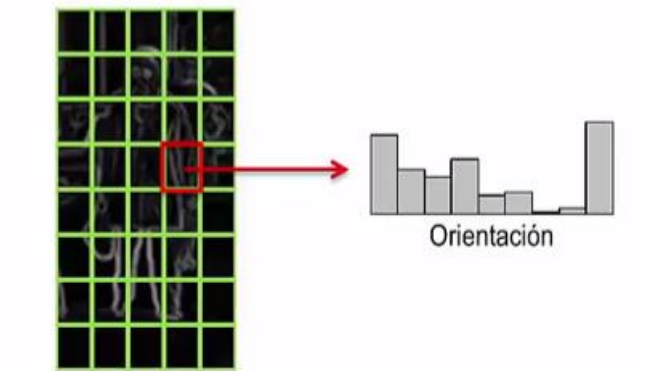


Figura 4-8: Histograma de gradientes orientados de cada celda [42]

Para ello, hay que tener en cuenta una serie de aspectos. Primero se debe seleccionar el tamaño de cada celda. Por lo general suelen ser de 6x6 o de 8x8 pixeles, en este caso pondremos el ejemplo de 6x6 (véase **Figura 4-9**).

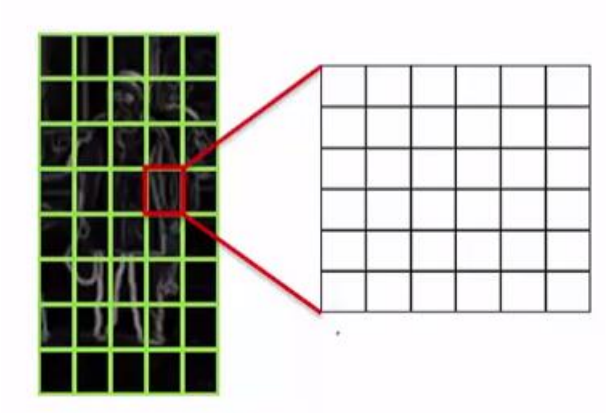


Figura 4-9: Tamaño de celda [42]

Otro aspecto a considerar es como se divide el rango de orientaciones en un número de intervalos fijo. Si se toma la orientación del gradiente con signo, este rango ira desde cero hasta 360º. Si por el contrario, se toma la orientación del gradiente sin signo, este rango irá desde cero a 180º. En este caso, pondremos este último como ejemplo (**Figura 4-10**), pues ya se verá más adelante que será el utilizado en este proyecto.

Figura 4-10: Rangos de orientación [42]

En este caso, dos gradientes con la misma dirección con sentidos inversos se consideran equivalentes.

Finalmente, el último parámetro a tener en cuenta es en cuántos intervalos se divide el rango de orientación. En este ejemplo como se ve en la **Figura 4-10**, se ha utilizado una división en 9 intervalos que suele ser también bastante habitual, y será, como veremos también más adelante, el utilizado para nuestro proyecto. Así, el rango ha quedado dividido en 9 sub-rangos: [0°, 20°), [20°, 40°)... [160°, 180°]).

Una vez fijados los parámetros anteriores, se procede a calcular cada uno de los gradientes en cada pixel de la celda, calculando para ello su orientación y magnitud, tal

como se ha explicado anteriormente. El resultado de estos gradientes puede verse en la **figura 4-11**:

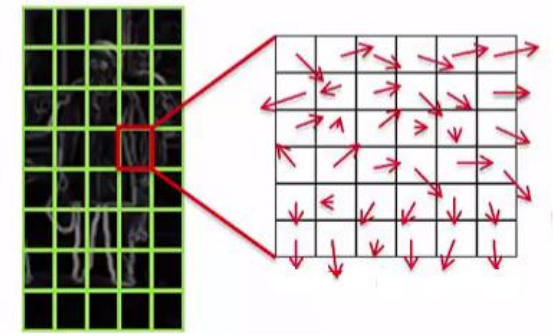


Figura 4-11: Gradientes de cada celda [42]

Una vez calculados, cada uno de estos gradientes quedará asignado a uno de los intervalos (**Figura 4-10**) en función de su orientación. De esta forma cada intervalo tiene asignados todos aquellos gradientes con una orientación dentro de los límites de dicho intervalo y cada uno de esos gradientes con una magnitud diferente.

Finalmente, el valor de uno de estos intervalos en el histograma final se obtiene acumulando la magnitud de los gradientes asignados a ese intervalo (**Figura 4-12**).

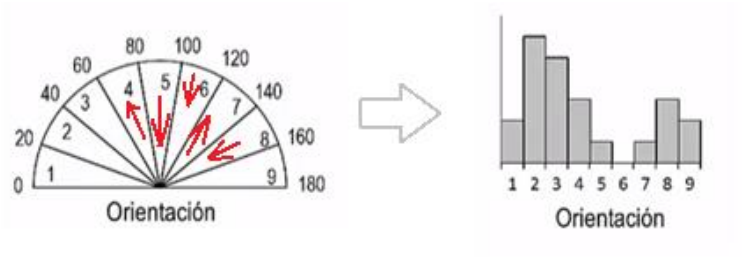


Figura 4-12: Obtención histograma final [42]

Este cálculo del histograma se puede formalizar matemáticamente de la siguiente manera:

$$h\ (k) = \sum_{(x,y)\in C} \omega_k(x,y) g(x,y) \qquad \textbf{(4.5)}$$

Dónde:

- h (k): el histograma en el intervalo k
- C: la celda donde se quiere calcular el histograma
- g(x,y): magnitud del gradiente
- wk(x,y): factor de asignación que sigue la siguiente expresión.

$$\omega_k(x,y)=\begin{cases}1 & \text{si } (k-1)\delta\theta \le \theta(x,y) < k\delta\theta \\ 0 & \text{en caso contrario}\end{cases}$$

Con esto, se podría realizar el cálculo de los histogramas de cada celda, concluyendo con ello, el primer paso.

2º Una vez obtenidos los histogramas de cada celda se combinan para obtener la representación global de toda la imagen en forma de vector de características [43] (Figura 4-13).

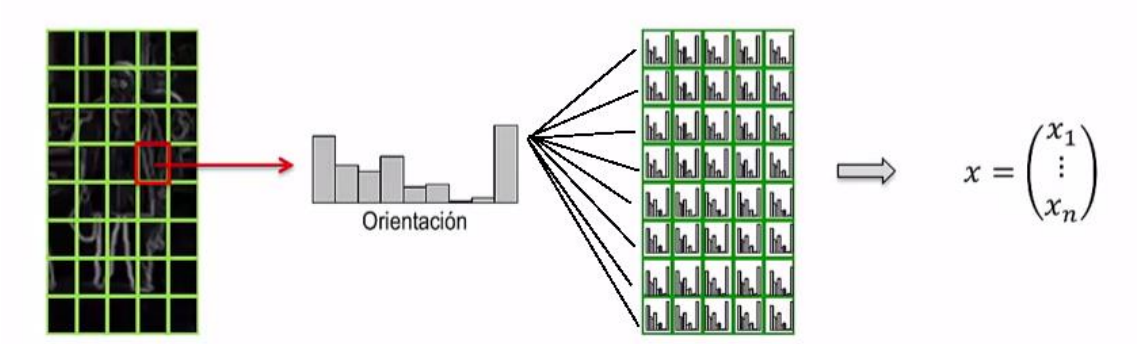


Figura 4-13: Obtención del vector de características [43]

Con esta combinación, se conseguiría un resultado como el mostrado en la siguiente figura (**Figura 4-14**):

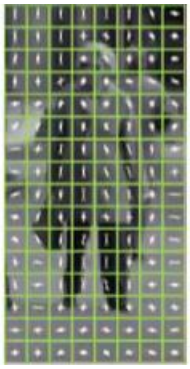


Figura 4-14: Resultado de HOG [41]

Una vez conseguido esto, debemos de asegurarnos que nuestro descriptor consiga la máxima invarianza posible a todas aquellas variaciones que se pueden producir en la imagen de entrada, ya sean iluminación, posición, escala, aspecto… Por ello, para evitar que los cambios de iluminación afecten al resultado de los histogramas obtenidos, se procede a normalizar los valores del histograma, de tal

forma que la magnitud global del gradiente sea similar en mismas imágenes con distintas iluminaciones.

Debido a que los cambios de iluminación en la imagen pueden ser distintos en cada zona de la imagen, se realiza una normalización adaptada en cada parte de la imagen. Para llevar a cabo dicha normalización, se agrupan las celdas del histograma en diferentes bloques (**Figura 4-15**).

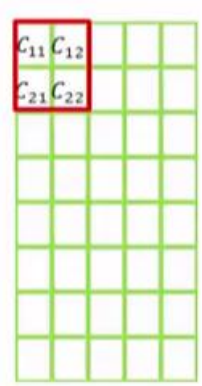


Figura 4-15: Bloques de celdas [43]

Así, para cada bloque se cogen los histogramas de cada celda y se concatenan para obtener el vector con la representación final del bloque (**Figura 4-16**).

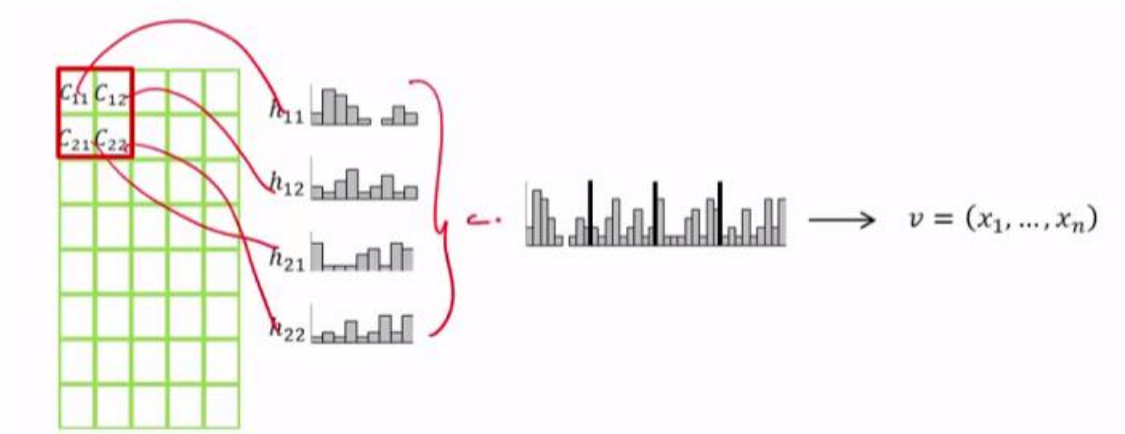


Figura 4-16: Obtención del vector del bloque [43]

Una vez conseguido los vectores de cada bloque se realiza su normalización, para ello podemos seguir las siguientes expresiones:

L2-norm: $$f = \frac{v}{\sqrt{\|v\|_2^2 + e^2}}$$ **(4.6)**

L1-norm: $$f = \frac{v}{(\|v\|_1 + e)}$$ **(4.7)**

L1-sqrt: $$f = \sqrt{\frac{v}{(\|v\|_1 + e)}}$$ **(4.8)**

L2-Hys: ésta se calcula tomando la L2-norm, se limitan los valores de v a 0.2 y después se vuelve a normalizar.

Dónde:

v : vector no normalizado que contiene todos los histogramas en un bloque dado.

$\|v\|_k$: K-norma del vector, donde k=1,2.

e : Constante de pequeño valor (se utiliza para evitar el 0 en el denominador).

En los siguientes apartados veremos con cuál de estas normalizaciones se consiguen los mejores resultados.

Los bloques a elegir, se definen de tal forma que tengan un cierto solapamiento entre ellos (**Figura 4-17)**, la redundancia que se va a obtener con este solapamiento va a ayudar a conseguir un descriptor más robusto ante deformaciones y variaciones en la forma del objeto. Por lo general, los bloques se colocan con una separación de una sola celda (tanto en horizontal como vertical) y la representación final del descriptor HOG se obtiene simplemente concatenando la representación normalizada de todos estos bloques solapados (**Figura 4-18**), finalizando con ello, la obtención del HOG.

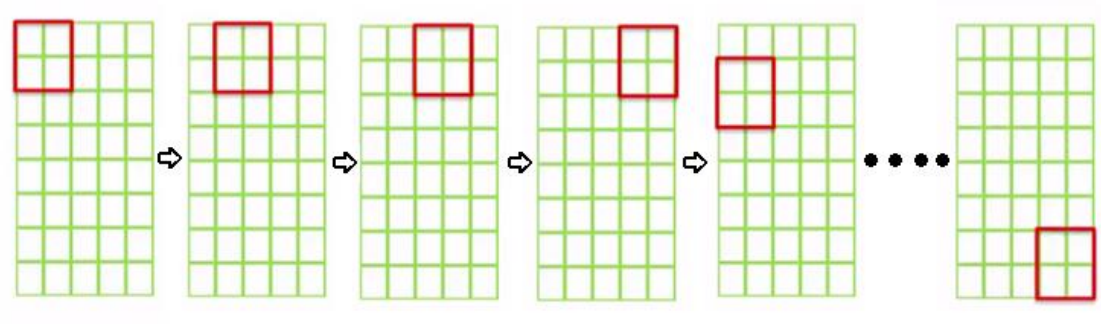


Figura 4-17: Solapamiento entre bloques [43]

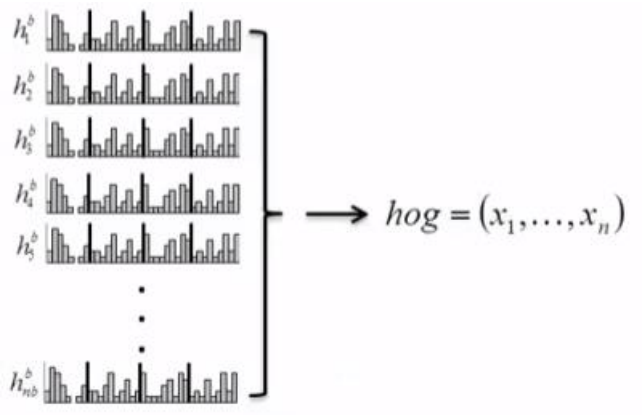


Figura 4-18: Concatenación de los histogramas de los bloques solapados [43]

4.1.2 Máquinas de Soporte Virtual (SVM)

Las Máquinas de Vectores Soporte son estructuras de aprendizaje basadas en la teoría estadística del aprendizaje [44]. Su función es transformar el espacio de entrada en otro de dimensión superior (infinita) para que el problema pueda ser resuelto mediante un hiperplano óptimo (de máximo margen).

Estos métodos están relacionados con problemas de clasificación y regresión. Dado un conjunto de ejemplos de entrenamiento (de muestras) podemos etiquetar las clases y entrenar una SVM para construir un modelo que prediga la clase de una nueva muestra. Intuitivamente, una SVM es un modelo que representa a los puntos de muestra en el espacio, separando las clases por un espacio lo más amplio posible. Cuando las nuevas muestras se ponen en correspondencia con dicho modelo, en función de su proximidad pueden ser clasificadas como pertenecientes a una u otra clase.

Dicho de otra forma, una SVM construye un hiperplano o conjunto de hiperplanos en un espacio de dimensionalidad muy alta (incluso infinita) que puede ser utilizado en problemas de clasificación o regresión. Para conseguir una clasificación correcta será necesario que exista una buena separación entre las clases.

La SVM busca un hiperplano que separe de forma óptima a los puntos de una clase de la de otra (**Figura 4-19**), que han podido ser previamente proyectados a un espacio de dimensionalidad superior.

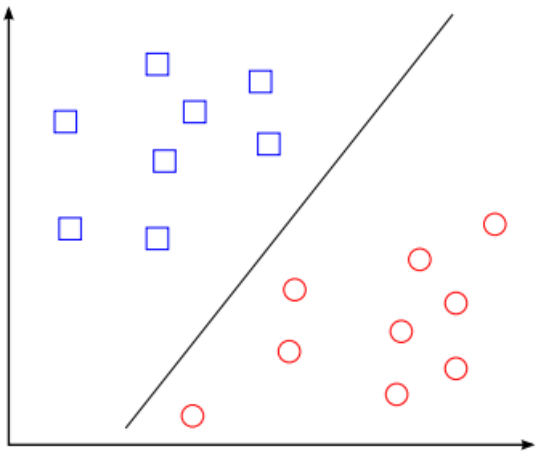


Figura 4-19: Hiperplano de separación de dos clases [45]

En ese concepto de "separación óptima" es donde reside la característica fundamental de las SVM: este tipo de algoritmos buscan el hiperplano que tenga la máxima distancia (margen) con los puntos que estén más cerca de él mismo. Por eso, también a veces se les conoce a las SVM como clasificadores de margen máximo. De esta forma, los puntos del vector que son etiquetados con una categoría estarán a un lado del hiperplano y los casos que se encuentren en la otra categoría estarán al otro lado.

Para separar linealmente los datos se procede a realizar un cambio de espacio mediante una función que transforme los datos de manera que se puedan separar linealmente. Esta función recibe el nombre de Kernel.

En nuestro caso, los conjuntos positivos y negativos corresponden con las clases "personas" y "no personas". Para ello, se necesita un entrenamiento previo de la máquina, introduciéndole ejemplos positivos (personas) y ejemplos negativos (no personas). Con todos los ejemplos de entrenamiento, el algoritmo de clasificación SVM crea una curva M-dimensional que divide ambos conjuntos, obteniendo de esta forma el kernel de la máquina. Las dimensiones del espacio dependen del número de componentes de cada vector a clasificar.

DESCRIPCIÓN MATEMÁTICA

Para una mejor comprensión de la máquina de soporte vectorial, se explicará su descripción matemática. Para ello, recurrimos a [3].

La formulación de las máquinas de vectores se basa en el principio de minimización estructural del riesgo. Mientras la mayoría de métodos de aprendizaje se centran en minimizar los errores cometidos por el modelo generado a partir de

ejemplos de entrenamiento (error empírico), el sesgo inductivo asociado a las SVMs radica en la minimización del denominado riesgo estructural. La idea es seleccionar un hiperplano de separación que equidista de los ejemplos más cercanos de cada clase para, de esta forma, conseguir lo que se denomina un margen máximo.

La idea es construir una función clasificadora que:

- Se encargue de minimizar el error en la separación de los objetos dados. Error en clasificación.

- Se encargue también de maximizar el margen de separación (mejora la generalización del clasificador).

Supongamos que tenemos el siguiente conjunto de entrenamiento:

$$(\bar{x}_i, z_i) : \bar{x}_i \in \Re^m, i = 1, .., N \qquad \textbf{(4.9)}$$

Supongamos también, que existe un hiperplano que separa los puntos de las dos clases. Los puntos **x** sobre el hiperplano satisfacen D(x) = $w^T x+b = 0$ **(Figura 4-20)** donde el vector **w** es normal al hiperplano, $||w||$ es la norma euclídea de **w** y $|b|/||w||$ es la distancia perpendicular del hiperplano al origen.

Sea D_+ *(D-)* la menor distancia del hiperplano de separación a la muestra positiva (negativa, respectivamente) más cercana. Definimos el margen del hiperplano (**Figura 4-20**) como la suma $D_+ + D$. En el caso linealmente separable, el algoritmo buscará simplemente el hiperplano con mayor margen. A continuación formularemos esta idea.

Ahora, supongamos que los datos de todo el entrenamiento satisfacen las siguientes desigualdades:

$$D(x) = w^T x_i + b > +1 \quad \text{para } z_i = +1 \qquad \textbf{(4.10a)}$$

$$D(x) = w^T x_i + b < -1 \quad \text{para } z_i = +1 \qquad \textbf{(4.10b)}$$

esto es,

$$z_i(w^T x_i + b) - 1 >= 0 \qquad \textbf{(4.10c)}$$

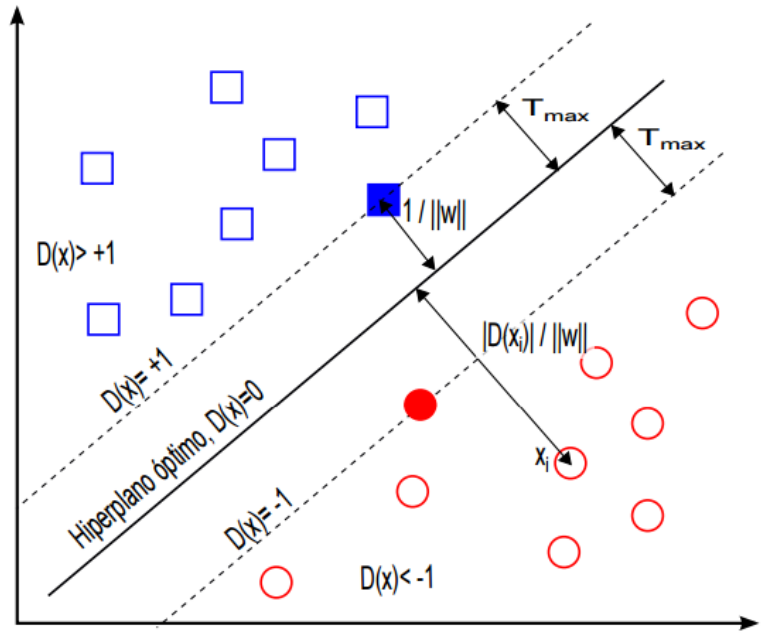


Figura 4-20: Hiperplano de máximo margen [45]

Ahora consideramos los puntos para los que se cumple la igualdad en (4.10a) (la existencia de este punto es equivalente a elegir una escala apropiada para **w y b**). Estos puntos están sobre el hiperplano H1*: D(x)= $w^T x_i$+b* = 1 con normal **w** y distancia al origen *|1-b|/||w||*.

De manera similar, la distancia hacia el origen del hiperplano H2 es *|-1-b|/||w||*. Por lo tanto, $D_+ = D = 1=||W||$ y el margen es simplemente *2/||w||*. Nótese que H1 y H2 tienen la misma normal, es decir, son paralelos, y que no hay puntos de entrenamiento entre ellos (**Figura 4-20**). Podemos, definitivamente, encontrar el par de hiperplanos que dan el máximo margen minimizando la función de coste *1/1 $||w||^2$* con las restricciones (4.10c).

Este problema de optimización con restricciones corresponde a un problema de programación cuadrático y es abordable mediante la teoría de la optimización [45].

Para ello, se pasará el problema a formulación lagrangiana. Esto nos permitirá sustituir las restricciones de (4.10c) por restricciones sobre multiplicadores de Lagrange, las cuales son más fáciles de utilizar.

Por otro lado, esta nueva formulación nos permitirá también que los datos del entrenamiento solo aparezcan en forma de productos escalares entre vectores. Esta propiedad es crucial para generalizar el procedimiento al caso no lineal como veremos.

Por lo tanto, introducimos *N* multiplicadores de Lagrange, a los cuales denotaremos por $\alpha_1, \alpha_2, \ldots, \alpha_n$, uno para cada una de las restricciones de (4.10c). La regla a seguir en aplicaciones lagrangianas es que para restricciones con la forma $\varsigma_1 >= 0$ las ecuaciones que las definen se multiplican por multiplicadores de Lagrange

positivos y se restan de la función objetivo con tal de formar el lagrangiano. En el caso de restricciones de la forma $ç_1 = 0$, los multiplicadores de Lagrange no tienen restricciones. Lo anterior nos da el lagrangiano:

$$L_p = \frac{1}{2}||\bar{w}||^2 - \sum_{n=1}^{N} \alpha_i z_i (\bar{w}^T \bar{x}_i + b) + \sum_{n=1}^{N} \alpha_i \qquad \textbf{(4.11)}$$

Ahora el objetivo es minimizar Lp con respecto a w, b y a la vez conseguir que las derivadas de Lp con respecto a los α_i se anulen, todo sujeto a las restricciones $\alpha_i > 0$ (restricciones $ç_1$). Esto significa que podemos resolver el problema de una forma equivalente el siguiente problema dual: maximizar Lp sujeto a la restricción de que su gradiente con respecto a w y b se anule, y sujeto también a la restricción de que $\alpha_i > 0$ (restricciones $ç_2$).

Al necesitar la anulación del gradiente de Lp con respecto a **w** y **b**, obtenemos las siguientes condiciones:

$$\frac{\partial L_p}{\partial \bar{w}} = 0 \Longrightarrow \bar{w} = \sum_{i=1}^{N} \alpha_i z_i \bar{x}_i \qquad \textbf{(4.12a)}$$

$$\frac{\partial L_p}{\partial b} = 0 \Longrightarrow \sum_{i=1}^{N} \alpha_i z_i = 0 \qquad \textbf{(4.12b)}$$

Dichas restricciones, las sustituimos en la siguiente ecuación:

$$p(x) = \frac{1}{m} \sum_{y_i=1+1} K(x, x_i) \qquad \textbf{(4.13)}$$

para obtener, finalmente:

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j z_i z_j \bar{x}_i \bar{x}_j \qquad \textbf{(4.14)}$$

La solución se obtiene minimizando Lp o maximizando LD.

Nótese que para cada muestra del entrenamiento existe un multiplicador de Lagrange α_i. Una vez conseguida una solución, aquellos puntos para los que $\alpha_i > 0$ se les denominará vectores soporte y serán los yacen sobre los hiperplanos H1, H2. El resto de las muestras tienen $\alpha_i = 0$, por ello, el vector **w** se escribirá como combinación de los vectores soporte.

Puesto que estos vectores son los más cercanos al hiperplano de separación, serán lo más difíciles de clasificar, y por tanto, deberían ser los últimos ejemplos a considerar a la hora de construir dicho hiperplano.

Puede observarse cómo **w** está determinado por el algoritmo de entrenamiento, pero no es este el caso del umbral **b**, aunque su obtención es inmediata.

Una vez que hayamos entrenado una SVM, para clasificar un patrón de evaluación **x** nos servirá con determinar en qué parte se encuentra de la frontera de decisión y asignarle la etiqueta de la clase correspondiente, es decir, asignamos a **x** la clase $sgnw^T x+b$ donde *sgn* es la función signo.

4.2 Desarrollo del algoritmo

4.2.1 Detección de las ROIs (Regiones de interés)

Para llevar a cabo la detección de peatones mediante el HOG y el SVM, primero se deben de determinar cuáles son las regiones de interés de las imágenes. Las regiones de interés son aquellas regiones de una imagen dónde cabe la posibilidad de que se localice un peatón. Es decir, los obstáculos pertenecientes a esa imagen.

En este caso, el procedimiento para encontrar peatones será detectar los obstáculos que hay enfrente del vehículo y, tras esto, que el HOG entrenado mediante el SVM determine si el obstáculo encontrado es un peatón o no.

Por ello, se deben de identificar las regiones de interés. Para llevar a cabo dicha identificación se ha utilizado el proyecto realizado por los profesores del departamento, llamado "*Automatic Obstacle Classification Using Laser Scanner and Camera Fusion*" [27] el cual está implementado en el IVVI 2.0.

Dicho proyecto se basa en la combinación de un láser escáner y una cámara para detección de objetos y clasificación. El láser es un "Sick LDMRS 4-layer Laser Scanner" y la cámara es una cámara trinocular "Bumblebee xB3" (**veánse figura 4-21 y figura 4-22**).

Figura 4-21 y Figura 4-22: Escáner láser (izquierda) y cámara trinocular (derecha) [46]

La idea del proyecto es utilizar la fusión de ambos elementos para conseguir una detección de obstáculos mejor que la que se conseguiría con la suma de las contribuciones individuales.

El escáner láser, es usado para la detección primaria de obstáculos y después para clasificación. La cámara por su parte, se utiliza su capacidad estéreo para la representación de una nube de puntos y para estimación de parámetros de alineación de datos. Después, una de las otras dos cámaras (recordemos que es trinocular) es utilizada como cámara para la captura de imágenes.

El funcionamiento general sería el siguiente: el láser escáner genera una nube de puntos, la cual representa algo de la realidad que hay en frente del vehículo. Los obstáculos forman parte de esa realidad y son localizados como concentraciones locales de puntos en dicha nube de puntos. Estos últimos, pueden ser categorizados matemáticamente como conjuntos (clusters). Estos conjuntos son extraídos y después son usados para la generación de las regiones de interés en las imágenes. Tras esto, las ROIs extraídas son procesadas para la clasificación de obstáculos utilizando métodos de inteligencia artificial aplicados a la visión por computador.

En nuestro caso, una vez extraída las regiones de interés, se procede a encontrar peatones en ellas calculando en dichas regiones las características HOG.

El hecho de identificar las regiones de interés nos supone una gran ventaja en cuanto al tiempo de computación. El tener que calcular las características HOG en la imagen completa supone un alto coste computacional. Por ello, si sólo nos centramos en sacar las características en partes concretas de la imagen (en las ROIs) conseguiremos reducir considerablemente el tiempo de procesamiento del algoritmo. Consiguiendo, con ello, una respuesta más rápida.

4.2.2 Histograma de gradientes orientados aplicado a la detección de peatones.

Para llevar a cabo la implementación del HOG, se utiliza como referencia el ejemplo “people detect” ofrecido en las librerías de OpenCV [6].

En dicho ejemplo se aplica el método explicado de Dalal y Triggs. A partir de un conjunto de imágenes positivas (es decir, con la figura de un peatón) y otro de negativas (con objetos que son “no peatones”) se obtiene la información necesaria para que después se lleve a cabo el entrenamiento.

En nuestro caso, el conjunto de imágenes positivas y negativas ha sido creado por nuestro equipo del departamento. Mediante la utilización de una cámara y un croma, se han realizado 17500 imágenes de tamaño de 64x128. De estas imágenes, 7000 de ellas son positivas y otras 10500 negativas. Cabe destacar que son imágenes variadas, es decir, los peatones presentan distintas alturas, anchuras, vestimentas y posiciones. Además, los objetos de las imágenes negativas también son distintos. A continuación puede verse un ejemplo de los conjuntos de dichas imágenes (**figuras 4-23** y **4-24**):

Figura 4-21: Conjunto de imágenes positivas

Figura 4-22: Conjunto de imágenes negativas

Estos conjuntos de imágenes se utilizaran para entrenar un detector mediante la Maquina de Soporte Vectorial (SVM). La información que obtendremos de las imágenes dará lugar a un descriptor HOG, el cual se utilizará después para detectar a los peatones en las imágenes que se desee.

Para comprobar el aprendizaje de la máquina, se comparará el primer modelo de detección obtenido con el conjunto de imágenes de entrenamiento. En el caso de que una imagen positiva no sea detectada, es decir, un falso negativo, esta deberá ser añadida al conjunto de muestra de positivos. Si, por el contrario, una imagen negativa es detectada como positiva, se tendrá un falso positivo y se tendrá que añadir dicha imagen al conjunto de muestras negativas. Este proceso se hará con todas las imágenes. Tras esto, se volverá a reentrenar tantas veces como sea necesario hasta tener un nivel de precisión en la detección aceptable.

Una vez hecho el entrenamiento, se dividirá las regiones de interés en celdas, y se calculará en ellas el histograma de gradientes. Para ello, se seguirán los pasos indicados en la descripción matemática del HOG, explicada en el apartado 4.1.1 y se fijarán a los distintos parámetros (tales como número de celdas, bloques, rango de orientación, sub-rangos…) los valores necesarios para conseguir los mejores resultados. Para la elección de estos valores se recurrirá a [1] y se explicará en el apartado 4.2.2.2.

4.2.2.1 Utilización de un Croma

En nuestro caso particular, a diferencia de otros proyectos, no se ha recurrido a una base de datos que pudiese encontrarse en internet para obtener las imágenes. Como se ha mencionado anteriormente, todas las imágenes utilizadas en el proyecto han sido hechas por el departamento de Sistemas y Automática. La particularidad de nuestras imágenes es que las imágenes positivas han sido hechas con un telón verde de fondo, es decir, un croma.

La razón de esto es que con un croma podemos obtener miles de muestras del objeto a clasificar (un peatón) en una sola sesión y luego entrenar con fondos aleatorios. Es decir, facilita la toma de muestras. Si quisiéramos, por el contrario, hacer la toma de muestras sin croma, tendríamos que hacer una gran cantidad de peatones en zonas urbanas, tanto en calles, en edificios… Esta tarea se hace tediosa, por lo que si utilizamos el croma podemos ahorrar mucho tiempo y además conseguir una variedad de imágenes mucho mayor.

Sin embargo, hay que tener en cuenta que no es conveniente entrenar el HOG sin insertarle un fondo a las imágenes antes. Es decir, no debemos de entrenar con el fondo verde. Recordemos que el método HOG se basa en la orientación de los gradientes para detectar un objeto en una imagen. Por tanto, al usar el croma no tenemos gradientes en los alrededores, haciendo que esto lleve a confusión. Por ello, habrá que insertar a la imagen fondos aleatorios, lo que supone también una ventaja,

ya que tendremos una variedad mayor de fondos que si hiciésemos las fotos en lugares concretos.

4.2.2.2 Estudio del rendimiento de HOG

En este apartado se explicará cómo afecta al rendimiento del HOG los diferentes parámetros utilizados. Para ello, se recurre al proyecto de Dalal y Triggs [1] donde se lleva a cabo un estudio en el que se justifica cuáles son los mejores valores a utilizar. Dichos valores son los que hemos utilizado en nuestro proyecto, tal y como se ha comentado anteriormente.

Para cuantificar el rendimiento del detector se trazan las curvas del Error de Equilibrio en la detección, es decir, Detection Error Tradeoff (DET) en una escala log-log, Miss Rate (grado de pérdida de candidatos) contra FPPW (en inglés, Falsos Positivos Por Ventana).

Se suele utilizar un Miss Rate de 10^{-4} FPPW como punto de referencia para los resultados. Esto corresponde a una tasa de error de alrededor de 0.8 falsos positivos en las imágenes.

En las curvas DET, pequeñas mejoras en el Miss Rate suponen grandes ganancias en FPPW. Por ejemplo, para un detector a 10^{-4} FPPW, cada 1% de reducción absoluta en el Miss Rate supone una reducción de FPPW en un factor de 1.57.

4.2.2.2.1 Escala de Gradiente

El rendimiento del detector es sensible a la manera en la que se calculan los gradientes, sin embargo los casos más sencillos son los que mejores resultados dan. Dalal y Triggs prueban a calcular los gradientes de diferentes maneras con tal de ver cuál es la mejor. Primero calculan dicho gradiente usando un suavizado gaussiano (Gaussian smoothing) el cual daña el rendimiento significativamente. Tras esto prueban a utilizar grandes máscaras, las cuales reducen también el rendimiento. Finalmente recurren a utilizar un filtro Gaussiano junto con una de las varias máscaras aplicables: Sobel 3x3, cúbica corregida, diagonales de 2x2, centrada y descentrada. Tras esto, llegan a la conclusión de que las máscaras centradas [-1,0,1] con σ=0 consiguen un mejor rendimiento (ver **Figura 4-25)** que todas las anteriores.

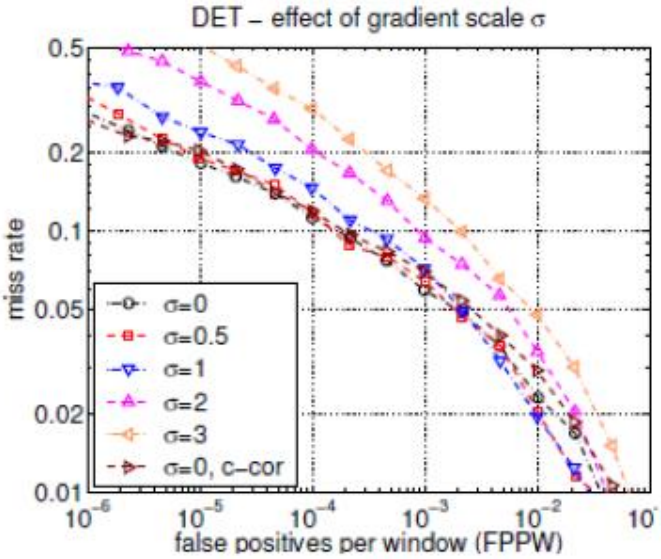


Figura 4-23: Estudio de los efectos que tienen sobre el rendimiento la modificación de la escala del gradiente [1].

4.2.2.2.2 Subrangos de orientación

Para calcular el HOG de una imagen, se divide el rango que contiene los valores de orientación de los gradientes en subrangos. Así, se tendrán en el histograma los gradientes de los píxeles agrupados en función del subrango al que pertenezcan. Dichos subrangos, afectan al rendimiento del HOG, por ello, Dalal y Triggs estudian cómo afecta a dicho rendimiento el aumento o la disminución de éstos.

Como muestra la **Figura 4-26** aumentar el número de subrangos de orientación mejora significativamente el rendimiento hasta llegar a 9 subrangos, a partir de ahí, no hay diferencia notable en dicho rendimiento. Esto es para el caso donde los subrangos están espaciados sobre los (0º - 180º), donde los signos de los gradientes son ignorados. Incluir el signo del gradiente (rango de orientación (0º-360º), como en el descriptor SIFT) disminuye el rendimiento. Incluso si se doblase el número de subrangos para conservar la resolución original de orientación también se tendría un peor rendimiento. Para la detección de personas, la amplia variedad de ropa y fondos hace que los signos no aporten información. Por tanto, Dalal y Triggs concluyen con que para la detección de personas el valor óptimo de subrangos es 9, tomando el rango de orientación de (0º-180º).

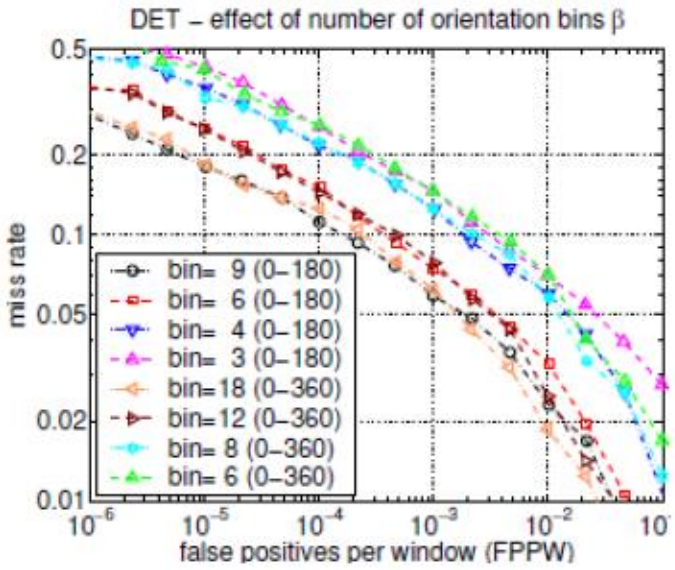


Figura 4-24: Estudio de los efectos que tienen sobre el rendimiento la modificación del número de subrangos de orientación [1].

4.2.2.2.3 Celdas Superpuestas

Otro de los parámetros que influyen en el rendimiento es el solape entre los bloques. Por ello, Dalal y Triggs realizan un análisis sobre el efecto de dicho solape, para valores de 0, 1/2 y 3/4 (**Figura 4-27)**. Tras esto, se llega a la conclusión de que los mejores resultados se obtienen con un solape de 3/4. El rendimiento mejora cerca de un 4% en 10^{-4} FPPW a medida que se aumenta la superposición desde 0 a 16. Esto se debe a que cuánto más espaciados estén los bloques, menos cantidad de ellos se necesitan para cubrir la imagen, lo que supone una mejora en el rendimiento.

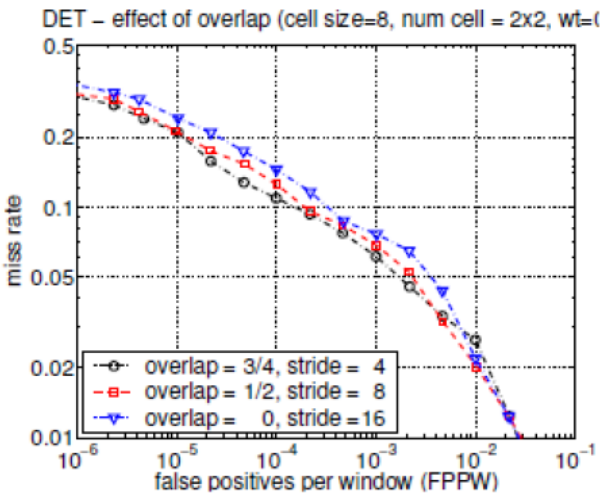


Figura 4-25: Estudio de los efectos que tienen sobre el rendimiento la modificación del número de celdas superpuestas [1].

4.2.2.2.4 Tamaño de los bloques y de las celdas

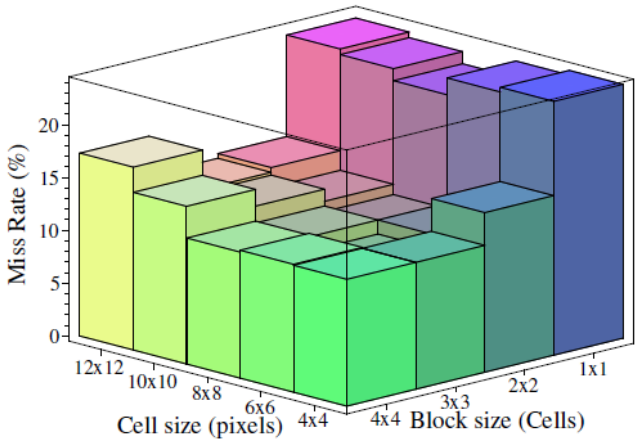


Figura 4-26: Gráfica que muestra la variación del rendimiento según el tamaño del bloque o la celda [1].

En la figura adjunta (**Figura 4-28**) se representa el Miss Rate de detección de personas en función del tamaño del bloque y del subbloque o celda. Se aprecia que para la detección de personas el valor óptimo se encuentra para un tamaño de bloque de 3x3 celdas y un tamaño de celda de 6x6 píxeles con un 10.4 % como Miss Rate en 10^{-4} FPPW.

A medida que aumenta el tamaño de los bloques los resultados empeoran debido a que también aumenta el tiempo. Por otro lado, en el caso de que los bloques sean muy pequeños también empeoran debido a que se suprime información importante.

4.2.2.2.5 Dimensiones de la ventana

Una ventana de detección de 64x128 incluye aproximadamente 16 pixeles de margen alrededor de las personas en los 4 lados de la imagen. La **Figura 4-29** muestra que esta frontera proporciona una cantidad significante de contexto que ayuda a la detección.

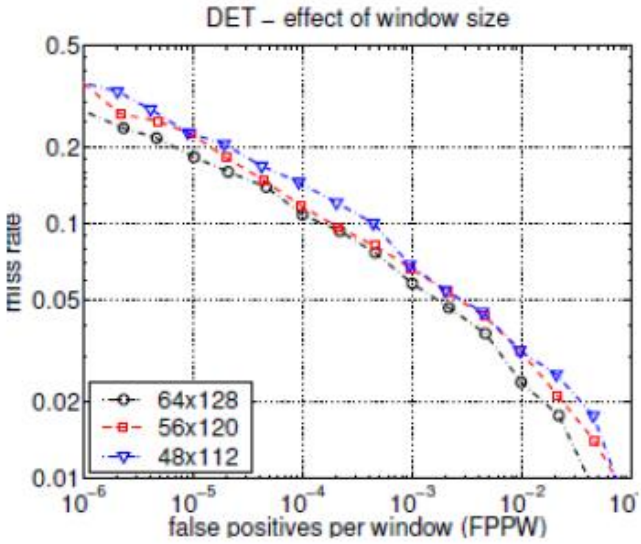


Figura 4-27: Estudio de los efectos que tienen sobre el rendimiento la modificación de las dimensiones de la ventana de detección [1].

En la **Figura 4-29**, se puede apreciar como al disminuir de 16 a 8pixeles (es decir, cambiar a una ventana de 48x112) disminuye el rendimiento en un 6% a 10^{-4} FPPW. Manteniendo por el contrario, la ventana de 64x128, pero incrementando el tamaño de la persona (disminuyendo de nuevo la frontera) se produce una perdida similar de rendimiento, incluso si la resolución de la persona ha sido incrementada.

4.2.2.2.6 Método de normalización empleado

Como se ha comentado anteriormente, para mejorar la invariancia a la iluminación y a los sombreados, se procede a normalizar el contraste del histograma calculado en cada uno de los bloques en los que se ha dividido la zona región a estudiar. Siendo, las posibles normalizaciones, las indicadas en la descripción matemática del apartado 4.1.1

La **Figura 4-30** muestra como utilizando las normalizaciones L2-Hys, L2-norm y L1-sqrt mostradas antes se obtiene prácticamente el mismo rendimiento, mientras que si se normaliza según el sistema L1-norm el rendimiento del proceso se reduce un 27% en 10^{-4} FPPW.

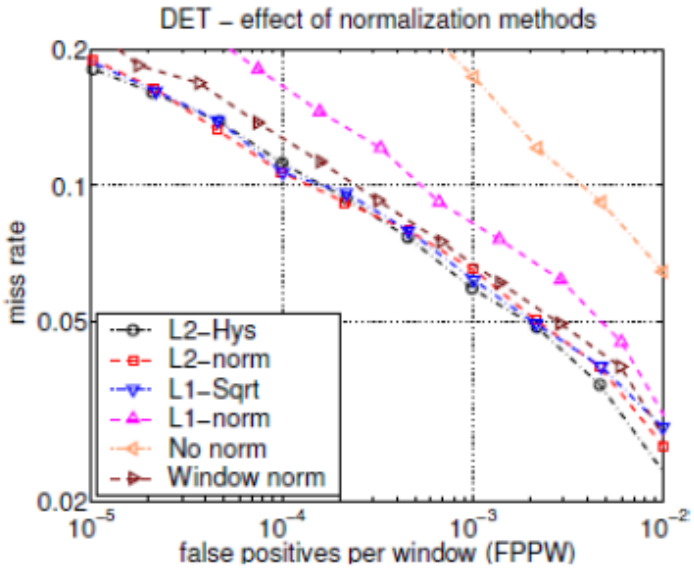


Figura 4-28: Estudio de los efectos que tienen sobre el rendimiento la modificación del método de normalización empleado [1].

4.2.2.2.7 Ancho del Kernel, gamma, en el kernel SVM

Dalal y Triggs utilizan por defecto un SVM lineal suave entrenado con SVM^{LIGHT} (ligeramente modificado para reducir el uso de memoria para evitar problemas con los vectores descriptores grandes). Usar, por el contrario, un kernel SVM Gaussiano (*exp* (*-gamma* $||x - x'||^2$)), aumenta el rendimiento alrededor de un 3% en 10^{-4} FPPW a costa de un mayor tiempo de computo (véase **Figura 4-31)**.

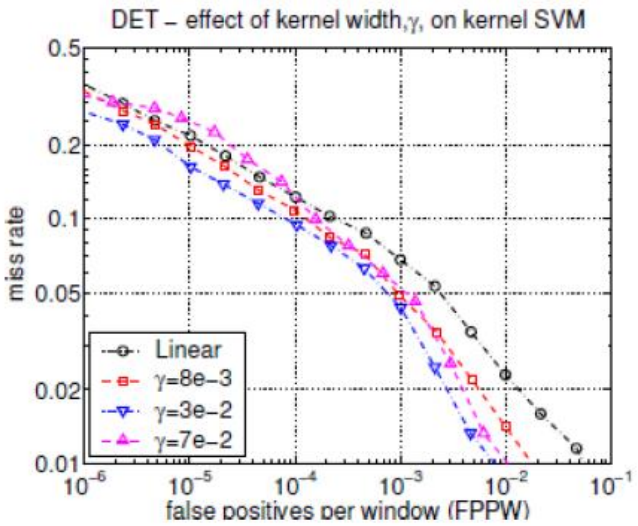


Figura 4-29: Estudio de los efectos que tienen sobre el rendimiento del ancho del kernel, gamma, en el kernel SVM [1].

Finalmente, Dalal y Triggs concluyen con que los valores estudiados anteriormente son los valores indicados con los que conseguir los mejores rendimientos.

De esta forma, demuestran que el método HOG presenta mejores resultados que métodos como los descriptores SIFT o los Wavelet Haar.

4.2.3 Entrenamiento de la SVM para la detección de peatones

Para llevar a cabo el entrenamiento de la SVM, necesitamos un conjunto de muestras positivas (con la figura de un peatón) y un conjunto de muestras negativas (con objetos que son "no peatones"). Con estos, como ya se ha comentado, el clasificador crea un espacio N-dimensional donde sitúa las muestras y las separa utilizando para ello un hiperplano de máximo margen.

Sin embargo, en función de la calidad de las imágenes (es decir, si presentan información útil o no) y de la precisión con la que se separen las muestras, se conseguirán mejores o peores resultados. Por ello, para comprobar las prestaciones del SVM, tenemos que estudiar una serie de parámetros para ver el rendimiento conseguido. Este estudio de parámetros se realizará en el apartado 6.

Además, para llevar a cabo el entrenamiento, primero se debe de hacer una serie de comprobaciones. Éstas se explican en los siguientes apartados.

4.2.3.1 Adecuación de la Base de Datos

Para llevar a cabo el entrenamiento de la SVM, primero se debe de comprobar que el conjunto de imágenes con el que se trabaja es adecuado. En nuestro caso, como ya se ha comentado anteriormente, ha sido nuestro departamento el encargado de crear la base de datos de las imágenes. En el caso de la detección de peatones, las imágenes de entrenamiento deben de ser de 64x128, puesto que como ya se ha demostrado, son las que mejores resultados nos dan.

Para ello, se han creado 7000 imágenes positivas y 10500 negativas destinadas al entrenamiento de la máquina. En un principio iban a ser todas de tamaños 64x128, sin embargo con tal de realizar un algoritmo mejor, se decidió añadir alrededor de 500 imágenes que no cumplieran con esas dimensiones. De esta forma, se añadirán una serie de funciones al programa que permitan redimensionar dichas imágenes de forma

proporcional y nos permita tener un código más completo. Dichas funciones serán explicadas en el apartado 5.1.1.

Una vez entrenado se debe de comprobar el rendimiento de la máquina, por lo que se utilizaran 480 imágenes de prueba donde se le pedirá a nuestro algoritmo que detecte los peatones presentes. Estas imágenes son de 1280 x960.

4.2.3.2 Adecuación del Kernel

Para realizar el entrenamiento de la SVM, se debe de seleccionar un kernel apropiado. Como ya se ha comentado anteriormente, para la detección de peatones se utiliza por defecto un kernel SVM lineal. En nuestro caso, se ha decidido utilizar dicho kernel, puesto que es el que menos coste computacional nos supone. Sin embargo, hay que tener en cuenta que existen más tipos de kernel, los cuales podemos utilizar para el entrenamiento del SVM.

Los kernel que podemos usar son los siguientes:

1- Lineal.
2- Polinómico $k(x,x')= (s*(x*x')+c)^d$.
3- Exponenciales $k(x,x') =exp(-gamma ||x - x'||^2)$.
4- De tangente hiperbólica $k(x,x') = tanh(s*(x,x')+c)$.

En ellos, se debe definir también cada uno de los parámetros, tales como el grado del polinomio, el valor de gamma, el valor de sigma… [47]

Como se puede observar, tenemos una amplia variedad de kernels, que además, varían también en función del parámetro a usar. Por ello, al tener una gran cantidad de posibilidades, no se puede decidir a priori que kernel es el mejor para determinadas aplicaciones. Por esta razón, basándonos en el estudio de Dalal y Triggs, hemos decidido utilizar el Kernel Lineal. Sin embargo, a continuación ponemos un ejemplo de cómo sería el entrenamiento con un kernel Gaussiano, el cual, según el estudio anterior, nos aumentaría el rendimiento en un 3% en 10^{-4} FPPW.

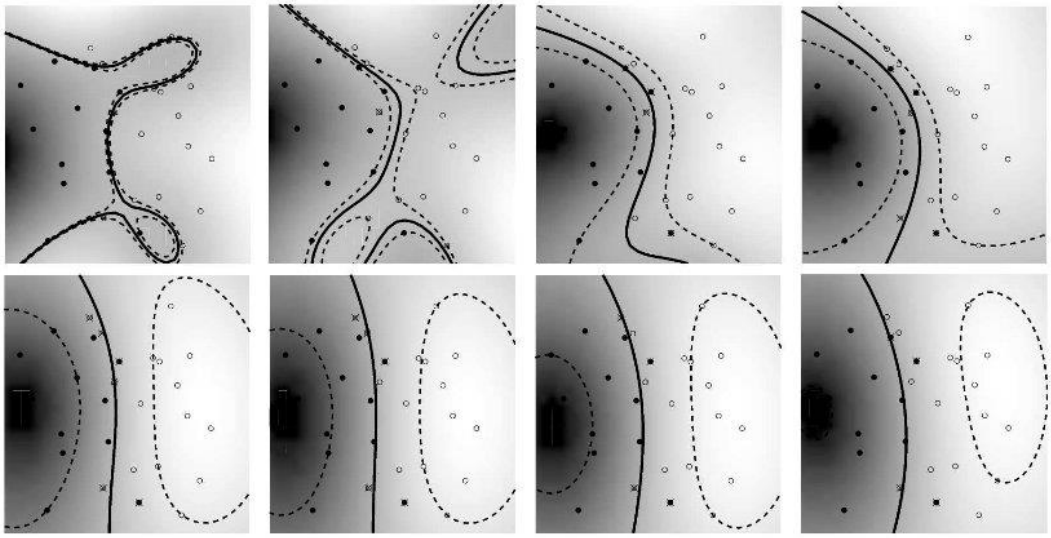


Figura 4-30 : Kernel gausiano para varios valores del parámetro gamma. Los valores de este parámetro van de gamma=0.1 (arriba izquierda) a gamma=0.8 (abajo derecha) [3].

En la **figura 4-32** se muestra un ejemplo de funcionamiento de un kernel gaussiano. A medida que aumenta gamma, más puntos se permiten encontrarse en la zona situada entre ambos conjuntos.

Así, tal y como se explica en [3]: "*Una vez trazada la línea que divide ambos espacios M-dimensionales, cuando queramos clasificar una nueva muestra no identificada deberemos introducirla en la máquina y representarla como un punto en este espacio. La distancia euclídea de la muestra a la curva creada en el entrenamiento resultará ser la medida que nos da la predicción sobre la muestra. El signo de esta magnitud nos informa que la muestra pertenece a "personas" o "no personas". Si el signo es positivo pertenecerá a la clase "persona" mientras que si es negativo pertenecerá a la clase "no persona". El módulo de la misma nos indica la probabilidad que hay de que una muestra pertenezca a un conjunto u otro. Por ejemplo, valores de 3.2 indicarán que la muestra pertenece al espacio de "personas" con una probabilidad mayor que si la predicción hubiera sido de 0.2. En el segundo caso, la máquina no predice con tanta certeza la pertenencia de la muestra a un conjunto u otro*."

4.2.3.3 Proceso de Realimentación

El proceso de realimentación es realizado para reducir el número de falsos positivos y negativos obtenidos con el primer modelo. Una vez obtenido el primer modelo, se prueba a detectar con el HOG entrenado los peatones que se encuentran en un conjunto de imágenes completas (imágenes de prueba). En el caso de que una imagen positiva no sea detectada, deberá ser añadida al conjunto de imágenes positivas. En el caso de que una imagen negativa sea detectada como positiva (falso

positivo) esta deberá ser añadida a la carpeta de imágenes negativas. Este proceso se hará con todas las imágenes de prueba.

Una vez terminado, se volverá a entrenar usando los conjuntos de imágenes positivas y negativas a los cuales se les han añadido las imágenes realimentadas. Con estos, se creará un nuevo modelo y se repetirá el test.

4.3 Desarrollo de un etiquetador

Como se acaba de explicar en los apartados anteriores, el entrenamiento realizado para detectar peatones en una imagen se lleva a cabo con varios conjuntos de imágenes (positivas y negativas). Tras esto, se procede a la parte de test, donde nuestro algoritmo deberá de buscar peatones en las regiones de interés y, en caso de encontrarlos, recuadrar dichos peatones e indicarnos sus posiciones.

Una vez recuadrados debemos de comprobar si dicha localización ha sido precisa. Para ello, tendríamos que etiquetar nosotros manualmente donde hay un peatón en una imagen y, después, comparar con las coordenadas indicadas por nuestro algoritmo. De esta forma, podríamos ver el nivel de precisión que tiene. Sin embargo, etiquetar manualmente todas las imágenes puede ser una tarea bastante tediosa. Por ello, para llevar a cabo el etiquetado de imágenes de una manera más rápida, se procede a desarrollar la lógica de un etiquetador.

El objetivo será realizar un código que, a partir de la posición inicial y final de un objeto, este sea capaz de calcular sus posiciones intermedias. De esta forma, no sería necesario etiquetar todas las imágenes manualmente, nos valdría con etiquetar la primera y la última de cada secuencia de vídeo. Hay que tener en cuenta que un vídeo es en realidad una secuencia de imágenes. Por tanto, tendremos una secuencia de imágenes donde sabremos la posición de un objeto en la primera imagen y en la última. Además también sabremos el número de imágenes intermedias.

Cabe decir también que la lógica realizada a continuación fue hecha con la intención de seguir el movimiento de bicicletas en lugar de peatones. En un principio el objetivo del proyecto era la detección de bicicletas, pero tras ver que éramos varios interesados en el proyecto nuestro profesor decidió (una vez empezado ya el proyecto) que uno de los alumnos se encargase de la detección de las bicicletas y el otro de peatones. De tal forma que así tendríamos una mayor variedad de aplicaciones. En mi caso, como ya se ha podido observar a lo largo del documento, me tocó realizar al final la detección de peatones. Es por ello que, aunque el proyecto entero está destinado a la detección de peatones, el siguiente apartado se explicará con bicicletas.

Nótese también que esto no afecta en los resultados. En este caso da igual que objeto tengamos, simplemente nos interesa calcular sus posiciones intermedias.

4.3.1 Consideraciones previas

Antes de comenzar con el cálculo matemático de las posiciones intermedias de las bicicletas, se deben tener en cuenta los siguientes aspectos:

- El eje de coordenadas a seguir no será el utilizado comúnmente. Como se va a utilizar el programa OpenCV, se deberán tener en cuenta las características de este programa a la hora de hacer los cálculos. En este caso, las coordenadas (0,0) de OpenCV se encuentran en la esquina superior izquierda.

- La posición de un objeto en una imagen está definida por las coordenadas (x,y) de la parte superior izquierda del objeto, además de su altura y anchura. Por tanto, cada vez que se haga referencia a las coordenadas de un objeto (en este caso bicicletas) se estará haciendo referencia a las coordenadas (x, y) de la parte superior izquierda de dicho objeto. Estas coordenadas, junto con la altura y la anchura definen una ventana que encuadra al objeto. A esta ventana se le denominará "Frame" y será la representación gráfica de la posición del objeto (ver **Figura 4-33**).

Figura 4-31: Ejemplo de Frame

- Además, se sabe que el número de imágenes intermedias es conocido, pero no siempre es el mismo. En función de la duración del vídeo, la cantidad de imágenes que lo componen será mayor o menor, y por tanto habrá más o menos imágenes intermedias. Por tanto, será necesario también introducir al programa el número de ventanas (Frames) intermedias que deberá de calcular.

Una vez tenido en cuenta los anteriores aspectos se empieza con la resolución del problema.

4.3.2 Desarrollo matemático

Se tiene una bicicleta en el instante inicial (Bo) y en el instante final (Bf). Se sabe la posición de esta en ambos instantes (coordenadas, altura y anchura) y se sabe también el número de imágenes intermedias (**Véase Figura 4-34**). En este caso, al tratarse de un ejemplo, se supone un número de tres imágenes intermedias, de tal forma que los cálculos sean más sencillos.

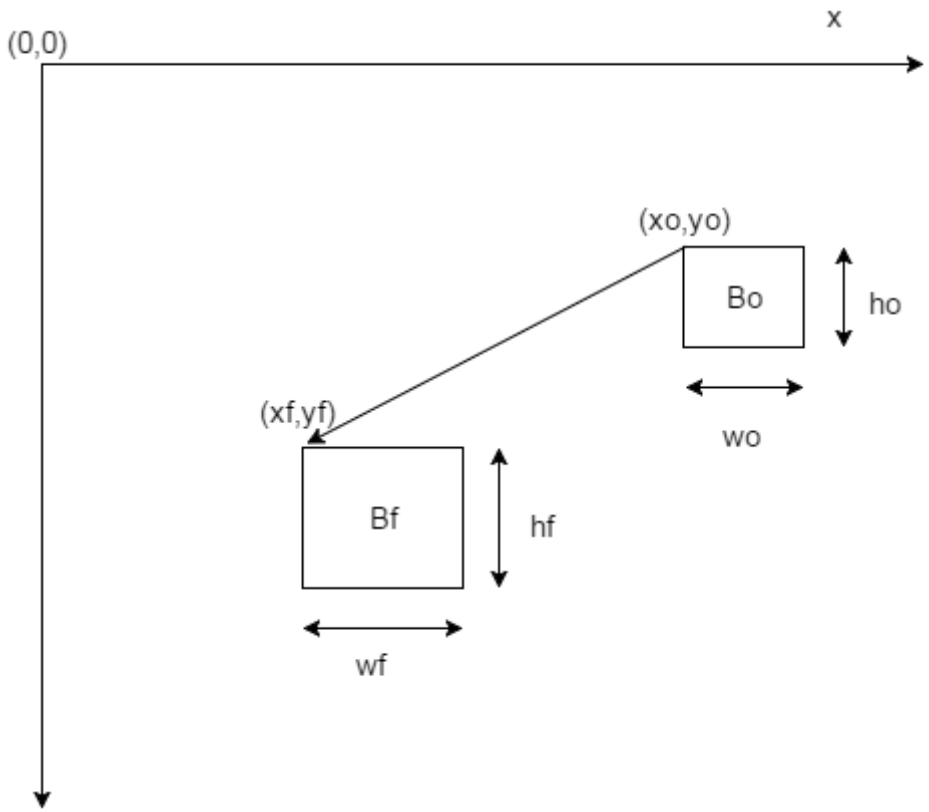


Figura 4-32: Representación gráfica del problema

Dónde:

- **Bo:** Bicicleta en el instante inicial.
- **Bf**: Bicicleta en el instante final.
- **xo, yo**: Coordenada x e y iniciales.
- **xf, yf**: coordenada x e y finales.
- **ho**: altura inicial del frame
- **hf:** altura final del frame
- **wo:** anchura inicial del frame
- **wf:** anchura final del frame

En la **figura 4-34** se ha representado de forma gráfica la posición de la bicicleta (inicial y final) así como su trayectoria. Para calcular las posiciones intermedias, se suponen las siguientes condiciones:

- El movimiento seguido por la bicicleta es recto (independientemente de la dirección). Al ser un cálculo matemático se debe de suponer que el movimiento siempre es el mismo, puesto que no se puede prever que la bicicleta cambie de dirección bruscamente.
- La velocidad de movimiento es constante (no hay aceleraciones ni deceleraciones).

A continuación se muestra la trayectoria seguida por la bicicleta. En ella se han dibujado las tres posiciones intermedias (**P1, P2, P3**) además de las posiciones inicial y final (**PO y PF)**.

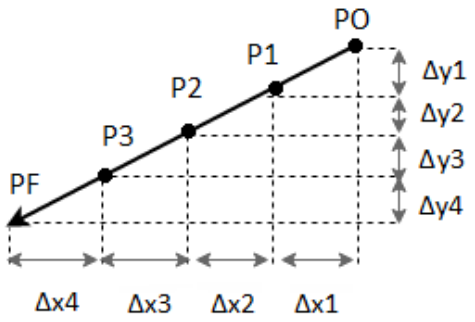


Figura 4-33: Posiciones y trayectoria de la bicicleta

Como se puede observar en la **figura 4-35**, el hecho de tener 3 Frames intermedios supone tener 4 incrementos en x e y. De la misma forma, si se tuviesen 4 Frames intermedios se tendrían 5 incrementos. Esto nos lleva a que la relación entre los frames y el número de incrementos en las coordenadas sigue la siguiente expresión:

$$\text{Numero de incrementos} = nf + 1 \qquad \textbf{(4.15)}$$

(Donde nf es el número de frames.)

Para calcular las coordenadas de todos los puntos intermedios, será necesario sumar al punto inicial un incremento en x y en y. Para ello, se deberá calcular dicho incremento.

Al estar suponiendo un movimiento constante, se deduce que todos los incrementos serán del mismo valor:

$$\Delta y1 = \Delta y2 = \Delta y3 = \Delta y4 = \mathbf{\Delta y} \quad \textbf{(4.16)}$$

$$\Delta x1 = \Delta x2 = \Delta x3 = \Delta x4 = \mathbf{\Delta x} \quad \textbf{(4.17)}$$

Por tanto, se obtiene que el valor de los incrementos en x e y para este caso sigue la siguiente expresión:

$$\Delta x = \frac{xo-xf}{4} \qquad \Delta y = \frac{yo-yf}{4} \quad \textbf{(4.18a)}$$

Dejándolo de forma general:

$$\Delta x = \frac{xo-xf}{nf+1} \qquad \Delta y = \frac{yo-yf}{nf+1} \quad \textbf{(4.18b)}$$

Una vez obtenidos los incrementos, se procede a calcular los puntos intermedios:

P1 = PO + Δx +Δy **(4.19a)**
P2 = PO + 2Δx + 2Δy **(4.19b)**
P3 = PO + 3Δx + 3Δy **(4.19c)**

Como se puede observar, todos los puntos siguen la siguiente expresión:

$$Pn = PO + n\Delta x + n\Delta y = PO + n * \frac{xo-xf}{nf+1} + n * \frac{yo-yf}{nf+1} \quad \textbf{(4.20a)}$$

$$\mathbf{Pn} = PO + \frac{n}{nf+1} * ((xo - xf) + (yo - yf)) \quad \textbf{(4.20b)}$$

De la misma forma con la altura y la anchura:

$$\mathbf{hn} = ho + \frac{n}{nf+1} * (ho - hf) \quad \textbf{(4.21)}$$

$$\mathbf{wn} = wo + \frac{n}{nf+1} * (wo - wf) \quad \textbf{(4.22)}$$

Con esto, se tendría las expresiones de las posiciones intermedias de las bicicletas. Dichas expresiones serán implementadas y añadidas a nuestro código.

Capítulo 5

5 Implementación del software

En este capítulo se pretende explicar al lector las partes del código realizado que se consideran de mayor importancia o complejidad, a fin de conseguir un mejor entendimiento por parte del mismo.

5.1 Descripción del código

Las partes del código más importantes son las relacionadas con el HOG y el SVM puesto que ambos algoritmos son la base del proyecto. Por ello, este capítulo se centrará en explicar las partes relacionadas con el entrenamiento y el test. Además, se hará también una pequeña descripción del código del etiquetador realizado por su utilidad en el proyecto.

5.1.1 Redimensionamiento de las imágenes

Como se ha indicado en capítulos anteriores, para llevar a cabo el proceso de extracción de características, las imágenes con las que se trabaja en el entrenamiento son de dimensiones 64 x 128 pixeles. Esto se debe a que la ventana de detección del HOG seleccionada tiene dichas dimensiones, puesto que son las que mejores resultados nos proporcionan [1].

Que las imágenes tengan las mismas dimensiones que la ventana de detección del HOG es fundamental, puesto que sino no se podrían extraer las características. Si se diese el caso de que alguna de las imágenes de entrenamiento (ya sean positivas o negativas) no tuviesen las dimensiones apropiadas, el código nos devolvería un error.

Como se mencionó en el capítulo 4.1.2 la base de datos utilizada es una base de datos propia creada por el departamento. La idea era generar todas las imágenes de entrenamiento con tamaño de 64x128 para evitar posibles fallos en la extracción de características. Sin embargo, con el fin de conseguir un algoritmo más completo, se decidió añadir dos funciones al código que se encargasen de redimensionar las imágenes positivas y negativas, de tal forma que incluso si alguna imagen no tenía las dimensiones indicadas, ésta pudiese ajustarse y usarse también en el entrenamiento. Por ello, se decidieron añadir 200 imágenes positivas de tamaños distintos a 64x128 y

otras 300 negativas, con el fin de probar dichas funciones. Las funciones encargadas de redimensionar las imágenes son *resize_imagePositive*() y *resize_imageNegative*().

Estas funciones se encargarán de buscar imágenes en los directorios de muestras positivas y negativas (respectivamente) y, una vez encuentren las imágenes llamarán a la funcion *resize* que se encargará de redimensionar las imágenes a 64x128 (véase **figura 5-1**).

```
resize (image, image_red, size,0.5,0.5,INTER_LINEAR);
```

Figura 5-1: Función resize

Donde:

- **image**: es la imagen de entrada.
- **image_red:** es la imagen de salida.
- **size:** tamaño al que se redimensionará la imagen.
- **0.5:** factor de escala del eje x.
- **0.5:** factor de escala del eje y.
- **INTER_LINEAR:** interpolacion bilineal (usada por defecto).

De esta forma, aunque por equivocación alguien introdujese una imagen que no cumpla las condiciones, ésta seria redimensionada evitando así los posibles errores. Una vez hecho esto, se procede a realizar las tareas de entrenamiento y test.

5.1.2 Código de entrenamiento

La parte del entrenamiento se considera importante debido a que los resultados obtenidos dependerán de cómo se haya realizado este entrenamiento. En los siguiente apartados explicamos los pasos que hemos seguido para llevar a cabo dicho entrenamiento.

5.1.2.1 Acceso a las carpetas de imágenes

La tarea del entrenamiento será encontrar y almacenar las características de todas las imágenes de las que disponemos para que después éstas puedan ser comparadas y se pueda realizar el test.

En nuestro caso, las imágenes de las que disponemos se almacenan en distintas carpetas, a las cuales nuestro algoritmo tendrá que acceder para poder sacar las

características mencionadas anteriormente. Dichas características serán guardadas en un archivo llamado "features.dat". Por su parte, se generará un vector descriptor que será guardado en "descriptorvector.dat" (este último se explicará más adelante).

```
static string posSamplesDir = "/home/jose/Documentos/Entrenamiento_SVM/Pictures/Positives/";
static string negSamplesDir = "/home/jose/Documentos/Entrenamiento_SVM/Pictures/Negatives/";

// Set the file to write the features to
static string featuresFile = "/home/jose/Documentos/Entrenamiento_SVM/Features.dat";
// Set the file to write the prediction to
static string PredictionFile = "/home/jose/Documentos/Entrenamiento_SVM/Predictions.dat";
// Set the file to write the SVM model to
static string svmModelFile = "/home/jose/Documentos/Entrenamiento_SVM/svm_lightmodel.dat";
// Set the file to write the resulting detecting descriptor vector to
static string descriptorVectorFile = "/home/jose/Documentos/Entrenamiento_SVM/descriptorvector.dat";
```

Figura 5-2: Dirección de las carpetas y archivos

Como se puede ver en la figura adjunta (**Figura 5-2**), los dos primeros directorios corresponden a las carpetas de imágenes positivas y negativas. Además pueden observarse también los directorios de las características y del vector descriptor mencionados anteriormente.

Utilizaremos los parámetros por defecto que no están incluidos en la clase HOG (**Figura 5-3**). La separación entre bloques es de 8x8. Los bloques son de 16x16 píxeles (1 bloque = 2x2 celdas, 1 celda = 8x8 píxeles).

```
static const Size trainingPadding = Size(0, 0);
static const Size winStride = Size(8, 8);
```

Figura 5-3: Parámetros no incluidos en la clase HOG.

5.1.2.2 Definición de los parámetros del HOG

Para realizar el entrenamiento hay que declarar un descriptor HOG al que se le asignan los parámetros por defecto. Tras esto, se le asigna el tamaño de ventana de 64x128 (véase **Figura 5-4**).

```
HOGDescriptor hog; // Use standard parameters here
hog.winSize = Size(64, 128);
```

Figura 5-4: Asignación de parámetros al hog

Siendo los parámetros del *HOGDescriptor* los siguientes:

- **win_size:** Tamaño de la ventana de detección.
- **block_size:** El tamaño de bloques en píxeles.

• **block_stride**: Paso Block. Tiene que ser un múltiplo del tamaño de la celda.

• **cell_size**: Tamaño de la celda.

• **nbins:** Número de contenedores.

• **win_sigma**: Parámetro Gaussiano para suavizar la ventana.

• **threshold_L2hys**: Umbral para el método de normalización.

• **gamma_correction**: Bandera para especificar si se requiere o no el procesamiento previo de corrección gamma.

• **nlevels**: Número máximo de incrementos de detección.

5.1.2.3 Train

En el Main del entrenamiento, una vez que se han redimensionado las imágenes, llamamos a la función *calculateFeaturesFromInput*(*currentImageFile, featureVector, hog*) para calcular el vector de gradientes de cada imagen. Como argumentos le pasamos el hog y el nombre de la imagen.

Una vez hecho esto se deben de extraer las características (**Figura 5-5**). Para ello, se utilizan los parámetros por defecto del HOG *winStride=Size(8,8)* y *padding=Size(0,0)* tal y como se comentó anteriormente. Para ello se utiliza la función *hog.compute*. Y tras esto, liberamos la imagen para no llenar la memoria.

```
vector<Point> locations;
hog.compute(imageData, featureVector, winStride, trainingPadding, locations);

imageData.release();
```

Figura 5-5: Paso para extraer las características HOG.

Una vez sacamos las características, se pasa al programa principal el vector *featureVector*, el cual contiene dichas características. Éstas, serán guardadas en el archivo que se creó al principio llamado feature.dat, el cual acaba ocupando un tamaño de 1,7 Mb. Tras esto, pasamos a la utilización del SVM.

SVM

En el programa principal, lo último que hacemos es pasar las características que hemos sacado con los métodos anteriores para realizar el entrenamiento. A partir de este punto, el algoritmo se centra en la detección de los objetos de la imagen y sus clasificaciones posteriores. En nuestro caso utilizamos un suave entrenador SVM lineal.

Primero, se definen los parámetros del SVM, tal y como se muestra en la **figura 5-6**:

```
// Set up SVM's parameters
    CvSVMParams params;
    params.svm_type    = CvSVM::C_SVC;
    params.kernel_type = CvSVM::LINEAR;
    params.term_crit   = cvTermCriteria(CV_TERMCRIT_ITER, 100, 1e-6);
```

Figura 5-6: Configuración de los parámetros del SVM

Dónde:

- **Svm_type:** es el tipo del SVM, se ha elegido el **CvSVM::C_SVM** puesto que es usado para la clasificación de n-clases, siendo 'n' mayor o igual a 2.
- **Kernel_type**: es el kernel utilizado. Como se ha mencionado anteriormente las funciones kernel se encargan de cambiar el espacio donde están las muestras para poder separarlas con un hiperplano lineal. En este caso se ha seleccionado el **CvSVM:LINEAR** que significa que no hay cambio de espacio, la separación de muestras se realiza en el mismo espacio creado por el SVM.
- **Term_crit**: es el criterio de terminación del algoritmo.

Estos parámetros se almacenan en el objeto "params" de la clase **CvSVMParams**.

Tras esto, se llama al método *train* para construir el modelo del SVM (véase **figura 5-7**).

```
// Train the SVM
CvSVM SVM;
SVM.train(trainingDataMat, labelsMat, Mat(), Mat(), params);
```

Figura 5-7: Función train

Este modelo será el que genere el vector descriptor que deberá ser cargado al hog para que pueda hacer distinción entre peatones y no peatones.

5.1.3 Código de Test

Una vez realizado el entrenamiento de las imágenes, se procede a la parte del test. En este apartado se analizarán las imágenes en las que se desean detectar personas y asi poder ver el grado de fiabilidad del entrenamiento generado.

Primero procedemos a entrenar el HOG mediante el SVM, empleando para ello la función *hog.setSVMDetector* (**Figura 5-8**). Sirve con realizarlo una única vez, por lo que se añade al *main*.

```
hog.setSVMDetector(descriptorVector);
```

Figura 5-8: Entrenamiento del HOG mediante el SVM

Como se puede ver en la imagen, se ha cargado el vector descriptor en el hog tal y como se comentó anteriormente. Recordemos que el vector descriptor es el que caracteriza la región correspondiente a un objeto en una imagen. Estos descriptores han sido evaluados por el clasificador, el cual determina la presencia o ausencia del objeto buscado, en este caso, el peatón. Por ello, ahora cuando el hog detecte un objeto, será capaz de comparar el descriptor de dicho objeto con el vector descriptor cargado y saber si se corresponde a un peatón o a un "no peatón".

Una vez entrenado llamamos a la función *detectTest,* a la cual le pasamos el HOG entrenado y la imagen donde queremos detectar a los peatones (**figura 5-9**).

```
detectTest( hog, aDetectar);
```

Figura 5-9: Función detecTest

Esta última, a su vez, llamará a las funciones *detectMultiScale* y *showDetections* (véase **figura 5-10**).

```
static void detectTest(const HOGDescriptor& hog, Mat& imageData) {

    vector<Rect> found;
    int groupThreshold = 2; // 2
    Size padding(Size(32, 32));
    Size winStride(Size(8, 8));
    double hitThreshold = -1; // tolerance -1
    hog.detectMultiScale(imageData, found, hitThreshold, winStride, padding, 1.05, groupThreshold);
    showDetections(found, imageData);
}
```

Figura 5-10: Funciones detectMultiScale y showDetections

Al llamar a la función *hog.detectMultiScale*, obtenemos como resultado un vector dinámico, al cual se accede en el caso de que se haya localizado en dicha región un peatón.

Para ello, se ajustan los parámetros de la función, que son los siguientes:

• **ImageData**: La imagen en la que queremos que detecte.

• **found**: Límites de lo objetos detectados.

• **hitThreshold**: Umbral para la distancia entre las características y el plano de clasificación de la SVM.

• **winStride**: El paso de la ventana. Debe ser un múltiplo de blockStride.

• **padding:** Parámetro Mock para mantener la compatibilidad de interfaz de la CPU (0,0).

• **scale**: Coeficiente de la ventana de detección de aumento.

• **group_threshold**: Coeficiente para regular el umbral de similitud. Cuando se detecta, algunos objetos pueden ser cubiertos por muchos rectángulos. 0 significa no realizar el agrupamiento.

Una vez hecho *detectMultiScale*, utilizamos la función *showDetections* para dibujar los rectángulos que encuadrarán a las personas que aparezcan en la imagen. A ésta, se le pasa el vector *found* y la imagen (**véase Figura 5-10**).

Y con esto tendríamos la prueba del algoritmo finalizada. Tras ello, habría que realizar la realimentación de falsos positivos y negativos, de tal forma que mejoremos los resultados.

5.1.4 Código del etiquetador

Como ya se ha comentado, para comprobar que la detección de los peatones con el HOG entrenado es precisa, se compararán las posiciones indicadas por éste último con las posiciones indicadas por nuestro etiquetador. Por ello, aunque la lógica del etiquetador es prácticamente matemática y sencilla de entender, se hará una breve descripción de su código, puesto que también es importante en el proyecto.

Recordemos que el etiquetador deberá de calcular las posiciones intermedias de un objeto, introduciéndole para ello su posición inicial, final y el número de posiciones intermedias.

Para ello, se han creado dos clases: una de ellas llamada "*Frame*", en la cual se declaran las coordenadas(x,y), altura y anchura que definen la posición de un objeto, y otra de ellas llamada "*Interpolation*". En esta última, se han creado las dos siguientes funciones: "*getInterpolation()*" y "*showInterpolation()*".

La función "*getInterpolation()*" se encargará de calcular las posiciones intermedias del objeto. Mientras, la función "*showInterpolation()*" se encargará de dibujar dichas posiciones en la imagen.

Una vez definido estas funciones, para llevar a cabo el cálculo de las posiciones intermedias simplemente habrá que introducir los datos necesarios y llamar desde el *main* a dichas funciones. El código completo del etiquetador, al igual que el del entrenamiento del hog, serán adjuntados en el **Anexo III.**

Capítulo 6

6 Resultados

En este capítulo se mostrarán los resultados obtenidos del proyecto realizado. Para evaluar dichos resultados se estudiarán una serie de parámetros, los cuales definiremos a continuación. Además también se hará un estudio de la precisión con la que nuestro algoritmo ha etiquetado a los peatones, utilizando para dicho estudio el etiquetador realizado.

6.1 Parámetros a estudiar

Para llevar a cabo el estudio de los resultados, primero se explicarán los parámetros que se utilizan para evaluar el rendimiento de la máquina. Estos son los siguientes:

- **Precision**: Proporción de True positives obtenidos en las detecciones. Se incluyen tanto los encontrados en las imágenes de prueba como en las imágenes de entrenamiento.

$$\textbf{Precision(\%)} = \frac{TP}{TP+FP}$$

- **Accuracy**: Proporción de detecciones correctas, tanto true positives como true negatives.

$$\textbf{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

- **MR** (Miss Rate): Indica el grado de pérdida de candidatos.

$$\textbf{MR} = \frac{FN}{Ntestpos}$$

- **Recall**: Relacionado con el grado de perdida de candidatos.

$$\textbf{Recall(\%)} = 100 - MR*100$$

- **FPPW:** Falsos positivos por ventana

$$\textbf{FPPW} = \frac{FP}{NtestNeg}$$

- Siendo:

- **TP:** “True Positives” Número de imágenes clasificadas como positivas siendo positivas.
- **FP**: “False Positives” Número de imágenes clasificadas como positivas erróneamente.
- **TN**: “True Negatives” Número de imágenes negativas clasificadas como negativas.
- **FN**:”False Negatives” Número de imágenes clasificadas como negativas siendo positivas.
- **Ntestpos**: Número de ejemplos de test positivos.
- **Ntestneg**: Número de ejemplos de test negativos.

Una vez teniendo esto en cuenta, procedemos a presentar los resultados en forma de tablas y gráficas.

6.2 Resultados obtenidos sin realimentación.

En este apartado explicaremos los resultados obtenidos con el primer modelo de entrenamiento, sin realimentar los falsos positivos y/o negativos obtenidos. Para realizar la comprobación, se han utilizado 480 imágenes de test donde nuestro hog entrenado se ha encargado de detectar a los peatones.

Con ello, se han obtenido los siguientes resultados de TN, TP, FN, TP:

FP	**3224**	**Falsos positivos totales**
FN	**0**	**Falsos negativos totales**
TP muestras	7000	True positives de las muestras
TP test	1386	True positives del test
TP	**8386**	**True positives totales**
TN muestras	10500	True negatives de las muestras
TNtest	0	True negatives del test
TN	**10500**	**True negatives totales**

Tabla 6-1: Resultados de TN, FN, TP y FP

En la tabla adjunta (**tabla 6-1**) pueden verse el número de falsos negativos y positivos así como el número de verdaderos positivos y negativos obtenidos. Consiguiendo, con ellos, los siguientes resultados:

	Resultados
Precision (%)	72,23
Accuracy(%)	84,45
FPPW	0,306
Recall(%)	100
MR(%)	0

Tabla 6-2: Resultados obtenidos

6.3 Resultados obtenidos con la realimentación realizada

Como se ha explicado a lo largo del documento, con tal de minimizar los fallos, cuando en una imagen se detecten falsos negativos y/o falsos positivos, estos deberán ser realimentados para el reaprendizaje de la máquina. Por ello, procederemos a realimentar los falsos positivos y negativos obtenidos en el primer modelo. Como puede verse en la **tabla 6-1**, el número de falsos positivos es 3224 y el número de falsos negativos es 0. A continuación mostramos los nuevos resultados obtenidos de TN, TF, FN, FP con dicha realimentación:

FP	**367**	**Falsos positivos totales**
FN	**137**	**Falsos negativos totales**
TP muestras	10224	True positives de las muestras
TP test	1405	True positives del test
TP	**11629**	**True positives totales**
TN muestras	10500	True negatives de las muestras
TNtest	0	True negatives del test
TN	**10500**	**True negatives totales**

Tabla 6-3: Resultados de TN, FN, TP y FP con la realimentación realizada

Como se puede observar en la tabla adjunta (**tabla 6-3**) el número de falsos positivos ha conseguido reducirse considerablemente hasta llegar a un valor de 367. Por contrapartida, el número de falsos negativos se ha visto aumentado en 137.

Con estos nuevos valores conseguimos los siguientes resultados:

	Resultados
Precision (%)	96,9
Accuracy(%)	97,94
FPPW	0,035
Recall(%)	98,66
MR(%)	0,0134

Tabla 6-4: Resultados obtenidos con la realimentación realizada

En este caso, hemos conseguido una precisión de 96,9 % y un valor de accuracy de 97,94%. Por otro lado, el número de falsos positivos por ventana se ha visto reducido hasta llegar a un valor de 0,0134. Además, los valores de MR y Recall se han visto ligeramente modificados. En el siguiente apartado realizaremos una comparación entre estos resultados y los anteriores.

6.4 Comparación de resultados

Ahora procederemos a comparar los resultados obtenidos con el primer modelo y los obtenidos tras la realimentación. Dichos resultados pueden verse en la **tabla 6-5**.

	Resultados sin realimentar	Resultados tras la realimentación
Precision (%)	72,23	96,90
Accuracy(%)	84,45	97,94
FPPW	0,306	0,035
Recall(%)	100	98,66
MR(%)	0	0,0134

Tabla 6-5: Comparación de resultados con y sin realimentación del entrenamiento

Como se puede ver en la **tabla 6-5**, una vez realizada la realimentación de los falsos positivos obtenidos en el primer modelo, se ha conseguido aumentar la precisión del nuevo modelo, consiguiendo con ello un valor de 96,9%. Es decir, se ha conseguido mejorar la precisión en un 24,67% más. Esto supone que el valor de los fasos positivos por ventana (FPPW) se ha visto reducido en un 0,271 menos. También puede observarse como el porcentaje de exactitud en la detección (accuracy) se ha visto incrementado. Alcanzando un 13,49 % más.

Por otro lado, se puede apreciar como el valor del Miss Rate se ha aumentado. Aunque el valor es pequeño, este supone que ha habido una pequeña proporción de peatones que no han sido detectados. El aumento en el MR, supone, por consiguiente una reducción en el Recall.

A continuación, con tal de apreciar mejor los cambios obtenidos con la realimentación, mostraremos un conjunto de ejemplos de imágenes con las detecciones realizadas antes y después de dicha realimentación:

IMAGEN 1 SIN REALIMENTACIÓN

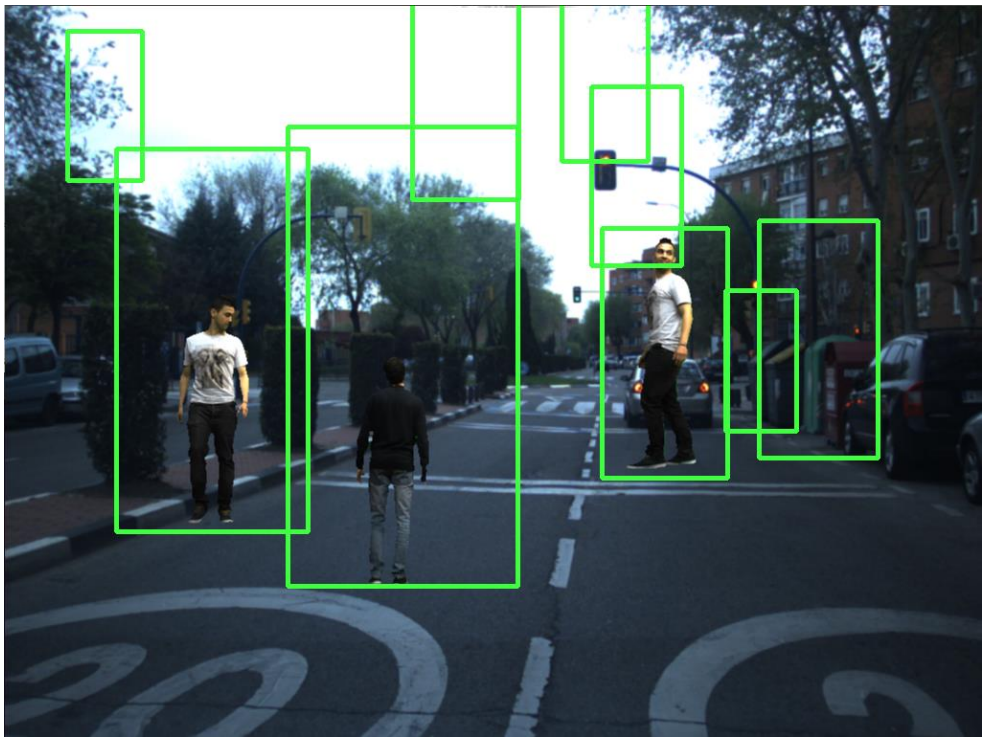


Figura 6-1: Ejemplo 1 de imagen sin realimentación

IMAGEN 1 TRAS REALIMENTACIÓN

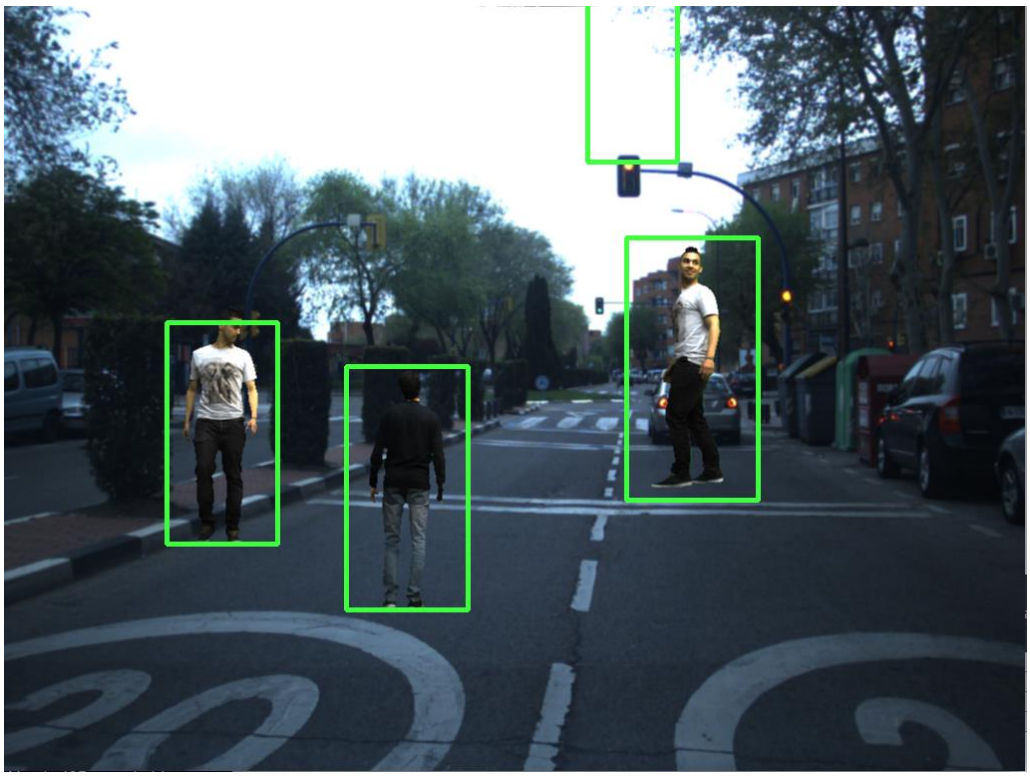


Figura 6-2: Ejemplo 1 de imagen con realimentación

IMAGEN 2 SIN REALIMENTACIÓN

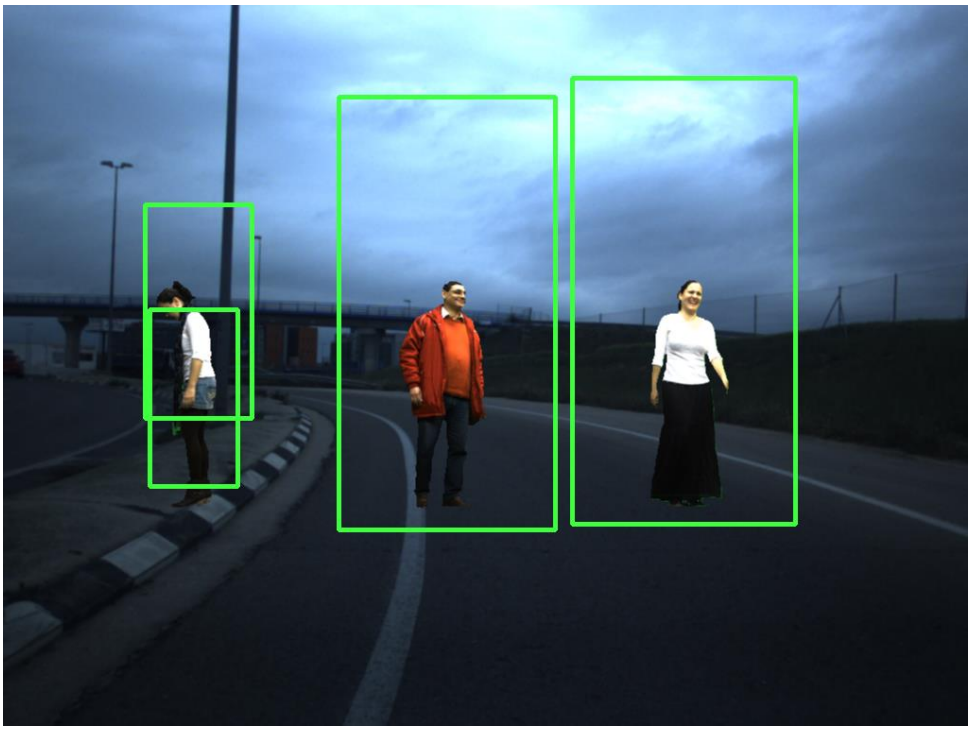


Figura 6-3: Ejemplo 2 de imagen sin realimentación

IMAGEN 2 TRAS REALIMENTACIÓN

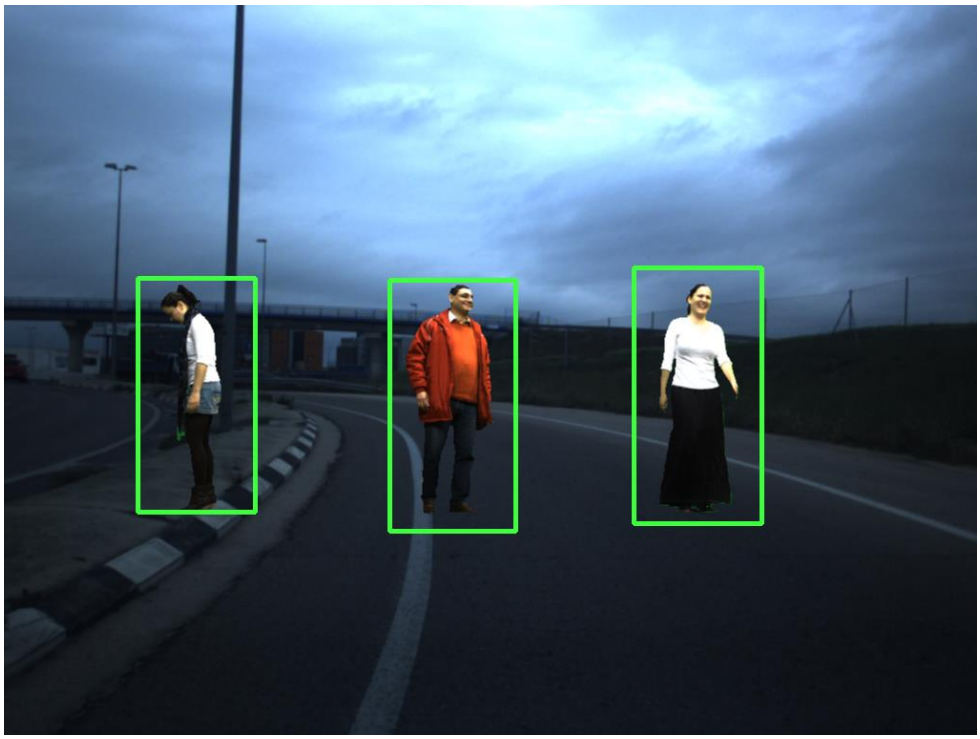


Figura 6-4: Ejemplo 2 de imagen con realimentación

IMAGEN 3 SIN REALIMENTACIÓN

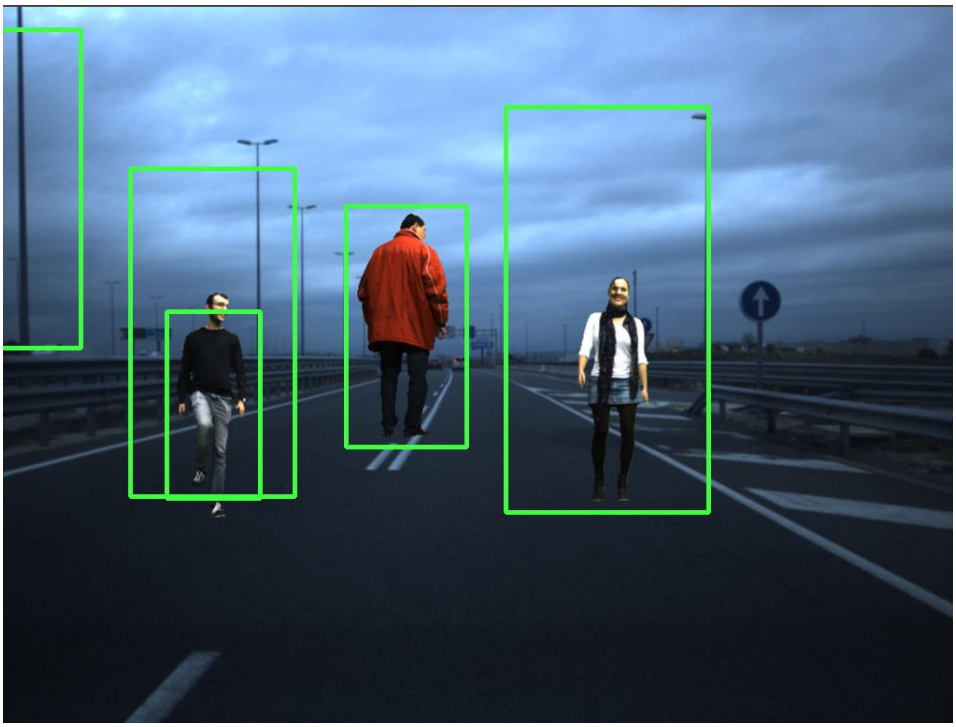


Figura 6-5: Ejemplo 3 de imagen sin realimentación

IMAGEN 3 TRAS REALIMENTACIÓN

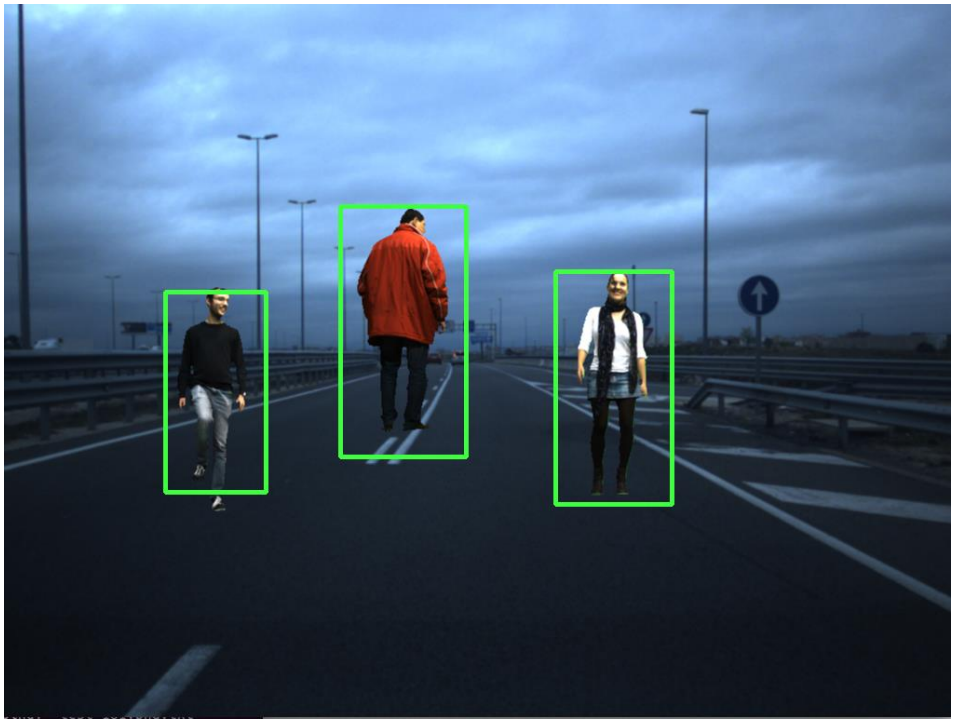


Figura 6-6: Ejemplo 3 de imagen con realimentación

IMAGEN 4 SIN REALIMENTACIÓN

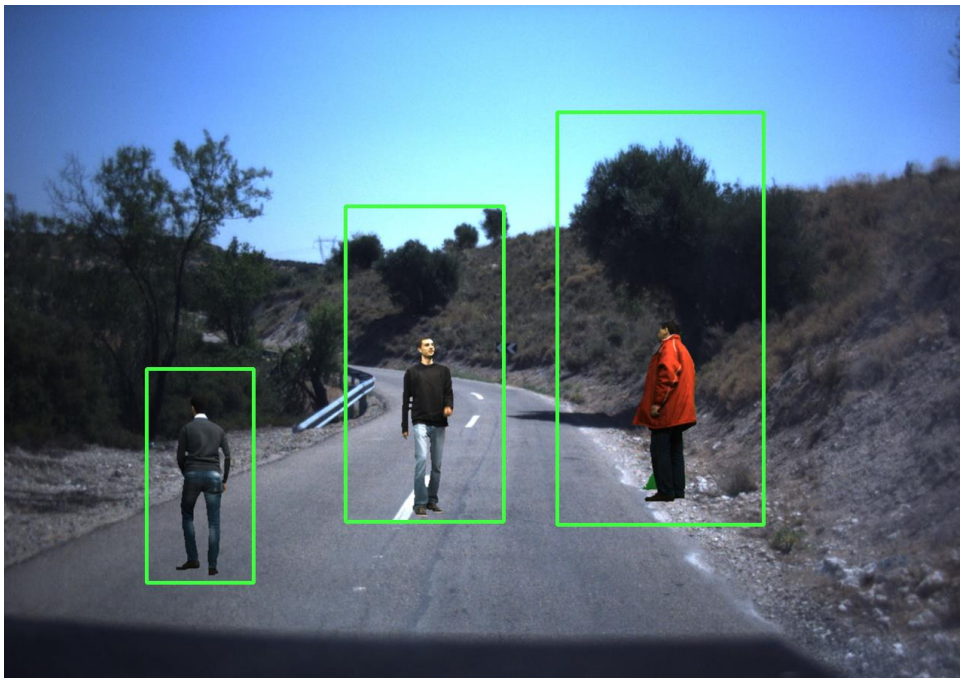


Figura 6-7: Ejemplo 4 de imagen sin realimentación

IMAGEN 4 TRAS REALIMENTACIÓN

Figura 6-8: Ejemplo 4 de imagen con realimentación

Como podemos ver en el conjunto de imágenes superior, una vez realizado la realimentación se ha conseguido disminuir el número de falsos positivos de manera considerable.

En la imagen 1 se ha conseguido pasar de 6 falsos positivos a un único falso positivo. En el caso de las imágenes 2 y 3 se puede observar cómo el número de falsos positivos se ha conseguido reducir hasta cero.

En el caso 4, sin embargo, aunque no tenemos ningún falso positivo, podemos ver cómo tras la realimentación un peatón que sí era detectado ahora no lo es. Esto supone un aumento en el número de falsos negativos. Aunque en este caso se ha empeorado el resultado, a nivel general los resultados son considerablemente mejores.

Además, como puede apreciarse en todas las imágenes, una vez realizada la realimentación, la etiquetación de los peatones es más precisa que sin dicha realimentación. En el último apartado veremos los resultados relacionados con dicha etiquetación.

6.5 Tiempo de cómputo

Como se ha ido comentando a lo largo del documento, uno de los objetivos del proyecto era conseguir que la detección de los peatones fuese lo más rápida posible. Por esta razón, es necesaria la localización de las ROI. Éstas, recordemos que han sido detectadas mediante el uso del láser escáner y la cámara trinocular.

En el caso de tener una secuencia de vídeo y analizar las imágenes de la secuencia enteras, es decir, aplicar hog en cada imagen completa, nos supondría un elevado coste computacional. Las imágenes captadas por la cámara tienen un tamaño de 640x480 píxeles, por lo que los píxeles a procesar en cada imagen de la secuencia serían de un valor de 307.200 pixeles. Como se puede intuir, este valor de pixeles es muy elevado y nos supondría un alto coste computacional. Esto supone un tiempo de computación de alrededor de 2,2 segundos.

Es por ello que debemos de identificar las regiones de interés y buscar los peatones en dichas regiones. Si en lugar de buscar los peatones en toda la imagen los buscamos en zonas concretas, se conseguiría reducir el coste computacional.

Sin embargo, hay que tener en cuenta un aspecto importante. Las ROI serán aquellas regiones donde se detecte un obstáculo, por ello, dependiendo de la secuencia de video habrá más o menos obstáculos, por tanto el número de pixeles a procesar no siempre va a ser el mismo. Por esta razón, no podemos decir un número

fijo de pixeles a procesar buscando en las ROI, pues irá variando con cada secuencia. Sin embargo, para hacernos una idea de la reducción de pixeles que conseguiríamos con la búsqueda en las ROI, decidimos utilizar una secuencia realizada en el laboratorio. En ella, el número medio de píxeles a procesar fue de 131.230 píxeles. Esto nos supone un tiempo de computación de aproximadamente 0,94 segundos. Como podemos observar, la reducción ha sido considerable. Sin embargo, volvemos a resaltar que este valor correspondería solo a esa secuencia. Si se diese el caso de que en otra secuencia el número de obstáculos es mayor, el tiempo de computación también lo será. De la misma manera si el número de obstáculos es menor, el tiempo computacional disminuirá.

Aun así, independientemente de la situación, vamos a conseguir siempre que la detección sea más rápida buscando en las ROI que buscando en la imagen completa. Por tanto, podemos decir que se ha conseguido reducir el coste computacional.

6.6 Resultados de la etiquetación

Finalmente, para terminar el estudio de los resultados, procedemos a mostrar la precisión con la que nuestro algoritmo ha etiquetado a los peatones. Para ello, como se ha comentado a lo largo del documento, compararemos las coordenadas indicadas por nuestro algoritmo por las coordenadas indicadas por nuestro etiquetador.

Tras realizar la comparación de todas las imágenes, obtenemos los siguientes resultados.

Media de error en la coordenada X:	**±6**
Media de error en la coordenada Y:	**±9**
Precisión en X:	**93%**
Precisión en Y:	**89%**

Tabla 6-6: Resultados en la etiquetación

El error en las coordenadas *x* e *y* hace referencia a la distancia entre la esquina superior izquierda de la ventana de detección del hog y la esquina superior izquierda de la ventana de detección del etiquetador.

Como podemos observar, la etiquetación realizada por el hog entrenado es muy parecida a la realizada por nuestro etiquetador. Consiguiendo en la etiquetación de la coordenada X una precisión de 93% y en la de la coordenada Y un 89%. Por tanto,

podemos afirmar que la etiquetación realizada por el hog entrenado es bastante precisa.

Sin embargo, hay que tener en cuenta también que los resultados pueden no ser del todo exactos. Aunque con el etiquetador no tenemos que etiquetar manualmente todas las imágenes sí tenemos que etiquetar las iniciales y finales de cada secuencia. Si la etiquetación manual de las posiciones iniciales y finales no ha sido del todo precisa, las coordenadas obtenidas en las intermedias tampoco lo serán. Por ello, a la hora de ver los resultados, se debe de tener en cuenta también los posibles errores humanos

Capítulo 7

7 Conclusiones y trabajos futuros

7.1 Conclusiones

En base a los resultados obtenidos se puede decir que se ha conseguido alcanzar el objetivo inicial del proyecto. Por un lado se tiene que nuestro algoritmo es bastante preciso en cuanto al número de aciertos, siendo el porcentaje de precisión de 96,9%. Además, el nivel de falsos positivos por ventana es bastante pequeño (0,035), lo que supone que el número de falsas alarmas en la detección será muy reducido. También cabe destacar que el valor del Miss Rate es bastante pequeño, teniendo un valor de 0,0134, lo que supone que prácticamente la totalidad de los peatones van a ser detectados.

Por otro lado, en cuanto a la precisión relativa a la etiquetación de peatones, también puede decirse que nuestro algoritmo es bastante exacto, siendo el máximo error de medida de aproximadamente ±6 en la coordenada X y ±9 en la coordenada Y.

Por último, cabe destacar también que el valor del tiempo de computación ha sido bastante bueno. Como se ha dicho a lo largo de todo el documento, se pretendía que la respuesta del algoritmo fuese lo más rápida posible de tal forma que se tuviese el mayor tiempo de reacción para evitar el accidente. Como se ha visto en los resultados, el tiempo medio de computación obtenida es de 0,94 segundos en la secuencia realizada, lo que nos supone una respuesta bastante rápida.

Por todo esto podemos concluir con que se han conseguido alcanzar prácticamente en su totalidad los objetivos deseados.

7.2 Trabajos futuros

Como posible trabajo futuro se propone bajar el número de falsos positivos y negativos obtenidos en el proyecto con tal de conseguir una respuesta más precisa del algoritmo. Para ello, se recomienda el uso de las técnicas boosting [35]. Las cuales, recordemos que se basan en la combinación de clasificadores débiles en cascada para conseguir un clasificador final que ofrezca buenos resultados. Esta técnica permitiría añadir más filtros a la clasificación haciendo que ésta tenga una mayor precisión. Sin embargo, se deberá de tener en cuenta la repercusión que esto puede tener en el coste computacional.

Capítulo 8

8 Costes del proyecto

En este capítulo se hará una estimación del coste del proyecto realizado. Por una parte se calculará el coste de las horas empleadas por el ingeniero para el desarrollo del proyecto y, por otra, se calculará el coste del material necesario para la elaboración de dicho proyecto.

En la siguiente tabla se muestra el número de horas empleadas por parte del ingeniero (**tabla 8-1**):

FASE	TIEMPO
Estudio y comprensión del proyecto	30 horas
Recopilación de documentación	60 horas
Desarrollo del algoritmo	100 horas
Prueba del algoritmo	40 horas
Desarrollo de la memoria	200 horas
TOTAL	**430 horas**

Tabla 8-1: Horas empleadas para el desarrollo del proyecto

Teniendo en cuenta que el coste por mano de obra es de 15€/hora (netos), tenemos que el coste total por el trabajo realizado es de:

430 horas * 15€/hora = **6.450€**

Por último, se muestra en la **tabla 8-2** el precio del material utilizado:

MATERIAL	COSTE
Pc	1.000 €
Paquete de librerías de OpenCV	Gratuito
Máquina virtual VirtualBox	Gratuito
IDE Qt Creator	Gratuito
Cámara (Modelo Bumblebee xB3)	3.000 €
Escáner láser (Sick LDMRS 4-layer)	4.000 €
Interfaz salpicadero	400 €
Otros materiales (cables, soportes…)	100 €
Croma	15€
TOTAL	**8.515€**

Tabla 8-2: Coste de los materiales del proyecto

Haciendo este último, junto con el coste de la mano de obra, un coste total de:

6.450€ + 8.515€ = **12.915 €**

9 Anexos

9.1 Anexo I: Recursos

En este apartado se listarán los recursos utilizados para el desarrollo del algoritmo realizado.

9.1.1 Tecnología

El proyecto ha sido desarrollado en la Universidad Carlos III de Madrid, en el departamento de Sistemas y Automática. Se ha construido sobre el sistema operativo Ubuntu [48] versión 14.0 LTS sobre un Lenovo z50. Se ha utilizado una máquina virtual llamada VirtualBox. Para llevar a cabo la obtención de las imágenes se ha utilizado una cámara Bumblebee xb3 y un Laser Sick LDMRS 4-layer. La implementación se ha realizado en C++ haciendo uso de la librería OpenCV [6]. La programación, tanto del método de HOG como el de SVM, ha sido facilitada gracias al IDE Qt Creator.

9.1.2 Software

9.1.2.1 Sistema Operativo

Para llevar a cabo el desarrollo de la aplicación aquí tratado se ha utilizado como sistema operativo Ubuntu. Ubuntu es un sistema operativo basado en GNU/Linux [48] y que se distribuye como software libre, el cual incluye su propio entorno de escritorio denominado Unity. Está orientado al usuario promedio, con un fuerte enfoque en la facilidad de uso y en mejorar la experiencia del usuario. Está compuesto de múltiple software normalmente distribuido bajo una licencia libre o de código abierto.

Instalación:

Para poder instalar Ubuntu en el dispositivo donde se vaya a trabajar, éste debe de cumplir con los siguientes requisitos mínimos:

- Procesador: x86 a 1 GHz.
- Memoria: 1 GB de RAM.
- Espacio en disco duro: 15 GB.
- Tarjeta gráfica: capaz de soportar una resolución de 800 x 600.
- Puerto USB.

- Tarjeta de red.
- Conexión a Internet no indispensable.

Estos requisitos corresponden a la versión 14.0.

En caso de no tener Ubuntu instalado se explica su instalación en el **Anexo II.**

9.1.2.2 OpenCV

OpenCV [6] es una biblioteca libre de visión artificial desarrollada por Intel en 1999. Se ha utilizado en infinidad de aplicaciones. Desde sistemas de seguridad con detección de movimiento, hasta aplicaciones de control de procesos donde se requiere reconocimiento de objetos. Esto se debe a que su publicación se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas.

Open CV es multiplataforma, existiendo versiones para GNU/Linux, Mac OS X y Windows. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras, visión estérea y visión robótica.

El proyecto pretende proporcionar un entorno de desarrollo fácil de utilizar y altamente eficiente. Esto se ha logrado realizando su programación en código C y C++ optimizados, aprovechando además las capacidades que proveen los procesadores multinúcleo. OpenCV puede además utilizar el sistema de primitivas de rendimiento integradas de Intel, un conjunto de rutinas de bajo nivel específicas para procesadores Intel

La versión utilizada en este TFG es la 2.3.1. En caso de no tener instalado OpenCV se puede llevar a cabo su instalación siguiendo las instrucciones indicadas en el **Anexo II.**

9.1.2.3 Qt Creator

Qt Creator [49] es un IDE multiplataforma, ideado por Trolltech para el desarrollo de aplicaciones creadas a partir de las bibliotecas de Qt. Esta bajo una licencia comercial y de código abierto.

Qt Creator, viene acompañado de un conjunto de herramientas para facilitar su uso. Además, posee un avanzado editor de código C++ (lenguaje utilizado en la realización de este proyecto) ya que es el lenguaje que utiliza de forma nativa.

Ademas, Qt Creator posee una herramienta gráfica (Qt Designer) diseñada para crear interfaces de usuario a partir de componentes de Qt.

En el caso de no tener instalado Qt Creator, solo debe de seguir los pasos indicados en el **Anexo**.

9.2 Anexo II: Instalación de Software

9.2.2 Instalación de Virtual Box

En el caso de que se desee trabajar en un ordenador que no tenga como sistema operativo Ubuntu, se puede instalar dicho sistema en una máquina virtual. Para instalar Virtual Box, debe de seguir los siguientes pasos:

1. Acceda a la página: https://www.virtualbox.org/ y haga click en "Download VirtualBox 5.0".

2. Seleccione que paquete de Virtual Box quiere descargar, en función de su sistema operativo. Por ejemplo, si su sistema operativo es Windows seleccione la opción "VirtualBox 5.0.24 for Windows hosts".

3. Una vez descargado, ábralo y pulse la opción "Instalar".

9.2.3 Instalación de Ubuntu

En el caso de no tener Ubuntu, es necesario seguir los siguientes pasos:

1. Vaya a la web de Ubuntu, seleccione la arquitectura que desee (23 o 64 bits) y pulse descargar.
2. Una vez descargado, seleccione el idioma deseado y después pulse "instalar Ubuntu".
3. Seleccione el tipo de instalación que desee.
4. Seleccione la zona horaria donde se encuentra.
5. Elija la distribución del teclado.
6. Por último, añada su usuario y contraseña.

En el caso de que el usuario desee instalar Ubuntu en Virtual Box como en este caso, deberá de seguir primero los siguientes pasos:

1. Abra VirtualBox y seleccione “Nuevo”. Una nueva ventana aparecerá
2. Elija el Sistema operativo que desee y arquitectura (32 vs 64 bit, Ubuntu, windows…)
3. Seleccione el tamaño de RAM
4. Seleccione el tamaño de almacenamiento y pulse el botón “Crear”.
5. Tras esto, seleccione START en virtualBox y una vez dentro realice la instalación de Ubuntu como se indicó arriba.

9.2.4 Instalación de OpenCV

Los pasos para instalar OpenCV son los siguientes:

1. **Instalar los paquetes necesarios:**

sudo apt-get install build-essential subversion cmake libgtk2.0-dev pkg-config zilb1g-dev libjpeg-dev libpng12-dev libjpeg62-dev libtiff4-dev libjasper-dev libunicap2-dev python-dev swig ffmpeg libdc1394-22-dev

2. **Obtener OpenCV**

Hay dos opciones para instalar OpenCV: bajar la última versión estable, o una versión anterior que se sepa seguro que funciona con nuestro entorno de desarrollo.

2.1 Última versión estable

Cree un directorio donde quiera instalar opencv y situese en él. Despues ejecute:

svn co https://opencvlibrary.svn.sourceforge.net/svnroot/opencvlibrary/tags/lates_tested_snapshot

Este comando dejará un directorio con nombre latest_tested_snapshot, que contiene la versión más reciente de opencv.

2.2 Versión anterior segura

Existe también la posibilidad de bajar alguna versión estable anterior, como por ejemplo la versión 2.3.1 de la dirección http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.3.1/OpenCV-2.3.1a.tar.bz2/download

3. **Instalación de OpenCV**

A continuación, nos situamos en el directorio opencv que está dentro del directorio de instalación, y creamos un directorio llamado *release*

Nos situamos en el directorio *release* recién creado y ejecutamos

cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D BUILD_PYTHON_SUPPORT=ON ..

A continuación ejecutamos

make

Cuando termine, ejecutar

sudo make install

Hay algunas variables de entorno que deben señalar al camino donde está instalado opencv.

Editamos el fichero .bashrc situado en nuestro directorio personal

cd

gedit .bashrc

E introducimos al final del fichero el texto siguiente:

export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH

A continuación, hacemos que se ejecute el fichero .bashrc con los comandos:

cd

source .bashrc

4. Configuración de compiladores para usar OpenCV

Para compilar los programas que generemos para openCV, tendremos que indicar al compilador algunos caminos.

Los averiguamos ejecutando:

pkg-config --cflags opencv

pkg-config --libs opencv

4.2 Qt

En el caso de utilizar OpenCV en Qt, habrá que incluir en el fichero .pro algo similar a:

INCLUDEPATH += /usr/local/include

LIBS += -L/usr/local/lib -lopencv_core -lopencv_imgproc -lopencv_highgui -lopencv_ml -lopencv_video -lopencv_features2d -lopencv_calib3d -lopencv_objdetect -lopencv_contrib -lopencv_legacy -lopencv_flann

9.2.5 Instalación de Qt Creator

Para instalar Qt Creator debe de seguir los siguientes pasos:

1. Visite la página de descargas de Qt http://qt.io/download e instale la versión que desee en función de su versión de Ubuntu. La instalación puede llevarla a cabo también el terminal:

   wget http://download.qt.io/official_releases/qt/5.0/5.0.2/qt-linux-opensource-5.0.2-x86-offline.run

2. Ajuste los permisos, corra el instalador y siga las instrucciones para completar la instalación:

   chmod +x qt-linux-opensource-5.0.2-x86-offline.run

   ./qt-linux-opensource-5.0.2-x86-offline.run

3. Instale g++ ejecutando el siguiente comando:

   sudo apt-get install build-essential

4. Instale las librerías OpenGL con el siguiente comando:

   sudo apt-get install mesa-common-dev

5. Fije la asociación de archivos. Cree un archivo llamado “Qt-Creator.desktop” y rellene dicho archivo con lo siguiente:

   [Desktop Entry]

   Version=1.0

   Encoding=UTF-8

   Type=Application

   Name=QtCreator

   Comment=QtCreator

NoDsiplay=true

Exec=(Install folder of QT)/Tools/QtCreator/bin/qtcreator %f

Icon=(Install folder of QT)/5.4/Src/qtdoc/doc/images/landing/icon_QtCreator_78x78px.png

Name[en_US]=Qt-Creator

6. Edite el archive llamado "defaults.list" en el mismo directorio. Añada la siguiente línea:

text/qtcreator=Qt-Creator.desktop;

7. Abra el archivo "mimeapps.list" y compruebe si la siguiente línea está presente:

application/vnd.nokia.qt.qmakeprofile=qtcreator.desktop

En el caso de que no lo este, añádala escribiendo el siguiente comando en el terminal:

sudo update-mime-database /usr/share/mime

9.3 Anexo III: Código del Algoritmo

A continuación se muestra el código total del algoritmo realizado, en primer lugar se mostrará la lógica del etiquetador t después, el entrenamiento. Todos ellos con comentarios añadidos para un mejor entendimiento por parte del lector.

CÓDIGO DEL ETIQUETADOR

main.cpp 1

```
//------------------------------Main.cpp------------------------------//


#include <iostream>
#include <string>
#include <math.h>
#include "frame.h"
#include "interpolation.h"

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv/cv.h>
#include <opencv/highgui.h>
#include <opencv2/opencv.hpp>


using namespace std;
using namespace cv;

int z,nf;


int main(){

   Frame inicial(55,90,262,373), final(567,179,56,161); // Posiciones inicial y final


    cout<< "Frame inicial\n";
    cout<< inicial;
    cout<< "Frame final\n";
    cout<< final;
    cout<< "Introduzca numero de frames\n";
    cin >> nf;                                  // numero de Frames intermedios

    Interpolation  A (inicial,final,nf);

    A.getInterpolation();
    A.showInterpolation();

    return 0;

}
```

```
//----------------------FRAME.h-----------------------//

#ifndef FRAME_H
#define FRAME_H

#include <iostream>
#include <string>

using namespace std;

class Frame{

    public:

    Frame();
    Frame(int x,int y,int w,int h);
    ~Frame();

    int getX()const;
    int getY()const;
    int getH()const;
    int getW()const;
    void set (int x, int y, int w, int h);

    Frame & operator = (const Frame &F);

    friend ostream & operator << (ostream &ver, const Frame &F);


    protected:

    int _x,_y,altura,anchura;

};


#endif // FRAME_H
```

```cpp
#include "frame.h"

using namespace std;

   Frame:: Frame(){ //Constructor vacio

    _x = 0;
    _y = 0;
    altura = 0;
    anchura = 0;

   }
   Frame:: Frame(int x,int y,int w,int h){ //Constructor parametrizado

       _x = x;
       _y = y;
       anchura = w;
       altura = h;
   }


   Frame::~Frame(){ // Destructor
   }

   Frame & Frame::operator = (const Frame &F){ // Constructor de Copia

       _x = F.getX();
       _y = F.getY();
       anchura = F.getW();
       altura = F.getH();
        return *this;
 }


   int Frame::getX() const{ //Funcion para obtener la X de un Frame

       return _x;

   }


 int Frame::getY()const{ //Funcion para obtener la Y de un Frame

    return _y;

}

 int Frame::getH() const{ //Funcion para obtener la H de un Frame

    return altura;

}

 int Frame::getW() const{ //Funcion para obtener la W de un Frame

    return anchura;

}

void Frame::set (int x,int y,int w, int h){ //funcion para asignar valores

    _x = x;
    _y = y;
   anchura = w;
   altura = h;
}

 ostream & operator << (ostream &ver, const Frame &F){ // Sobrecarga de cout
```

```
    ver<< F.getX()<<" "<< F.getY()<<" "<< F.getW()<< " "<< F.getH()<<"\n";
    return ver;

}
```

```
//--------------------INTERPOLATION.h-------------------------//


#ifndef INTERPOLATION1_H
#define INTERPOLATION1_H


#include <iostream>
#include <string>
#include "frame.h"
#include <vector>

using namespace std;

class Interpolation :public Frame{

    public:

    Interpolation ();
    Interpolation (Frame o,Frame f, int nf);
    vector<Frame> getInterpolation();
    void showInterpolation();

    private:

    Frame _o,_f,medio;
    int _nf;

    vector<Frame> medios;


};

#endif // INTERPOLATION1_H
```

```
//------------------Interpolation.cpp-------------------------------------------//

#include "interpolation.h"
#include "frame.h"
#include <iostream>
#include <fstream>

#include <vector>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/opencv.hpp>

using namespace cv;


Interpolation :: Interpolation (Frame o,Frame f, int nf){


    _o = o;
    _f = f;
    _nf = nf;

// Definimos los arrays donde guardar los valores de x,y altura y anchura:
   double x[100]={};
   double y[100]={};
   double h[100]={};
   double w[100]={};


ofstream fs("/home/jose/Documentos/original2:242-259.txt"); // fichero donde meter los
valores de (x,y, x+anchura, y +altura)

    int z = nf + 1; //Para simplificar la formula obtenida en los calcuos matematicos


      for (int n=1; n<z; n++){

   double j = double (n)/double (z); //lo mismo que z, para simplificar la formula

 //nos queda P[n] = Po - j*(Po-Pf), aplicado para cada caso:


               x[n] = o.getX() + j*(-o.getX() + f.getX()); //Cogemos la coordenada x del
frame inicial y el final y con ella hacemos los calculos.
               y[n] = o.getY() + j*(-o.getY() + f.getY()); //Cogemos la coordenada y del
frame inicial y el final y con ella hacemos los calculos.
               w[n] = o.getW() + j*(-o.getW() + f.getW()); //Cogemos la anchura w del
frame inicial y el final y con ella hacemos los calculos.
               h[n] = o.getH() + j*(-o.getH() + f.getH()); //Cogemos la altura h del
frame inicial y el final y con ella hacemos los calculos.

               fs << x[n]<<","<< y[n]<<","<< w[n] + x[n]<<","<< h[n] + y[n]<<endl; //
Guardamos los valores en el fichero


               Mat inicia = imread("/home/jose/Documentos/Entrenamiento_SVM/Pictures/
test_006.png"); //Imagen Original

               namedWindow("Imagen Original");

               imshow("Imagen Original", inicia);

               rectangle(inicia, Point(x[n], y[n]), Point(x[n]+ w[n], y[n] + h[n]),
CV_RGB(0,255,255)); //para ver sobre la marcha como avanza el rectangulo en la imagen

               namedWindow("Interpolaciones");

               imshow("Interpolaciones", inicia);//Imagen donde mostrar las
interpolaciones
```

```
                medio.set (x[n],y[n],w[n],h[n]); //Guardamos los valores en un Frame

                medios.push_back(medio); // Cargamos el Frame a un vector

                     waitKey(0);

      }

      fs.close();

}

vector<Frame> Interpolation::getInterpolation(){ // Para conseguir la interpolacion


        return medios;

}

void Interpolation::showInterpolation(){ // Para mostrar los valores de las
interpolaciones, x,y, h, w

         for(int n = 0; n < medios.size();n++){

         cout << "Frame numero  "<< n + 1 << "\n";
         cout << "x"<<"    "<<"y"<<"   "<<"w"<<"  "<<"h"<<"   "<<"\n";
         cout << medios.at(n)<<"x + w= "<<medios.at(n).getX()+medios.at(n).getW() <<";
"<< "y + h=" <<medios.at(n).getY()+medios.at(n).getH()<<"\n";


        }


}
```

CÓDIGO DEL ENTRENAMIENTO

main.cpp 1

```
#include <iostream>
#include <math.h>
#include<stdio.h>
#include<vector>

#include <time.h>
#include <stdlib.h>

//LIBRERIAS OPENCV
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/ml/ml.hpp>

//LIBRERIAS BOOST
#include <boost/filesystem.hpp>
#include "boost/filesystem/operations.hpp"
#include "boost/filesystem/path.hpp"
#include <boost/lexical_cast.hpp>

#include<sstream>

#include <fstream>


#include "opencv2/objdetect/objdetect.hpp"


#include <stdio.h>
#include <string.h>
#include <ctype.h>



#include <stdio.h>
#include <dirent.h>
#include <ios>
#include <fstream>
#include <stdexcept>



#include <QCoreApplication>
#include <iostream>
#include <math.h>
#include <stdio.h>
#include <vector>
#include <time.h>
#include <stdlib.h>


using namespace std;
using namespace cv;
namespace fs=boost::filesystem;

using std::vector;
using std::cout;
using std::endl;

bool save_box = false;

Mat image;
Mat temp;
Mat image_red;
//Size size (64,64);
Size size (64,128);


fs::path imagenADetectar="/home/jose/Documentos/Entrenamiento_SVM/Pictures/test/
test_081.png";
```

```
fs::path directorioP("/home/jose/Documentos/Entrenamiento_SVM/Pictures/P");
fs::path directorioN("/home/jose/Documentos/Entrenamiento_SVM/Pictures/N");


fs::path redimensionadoP ("Positives");
fs::path redimensionadoN ("Negatives");
fs::path nombreOriginal;
fs::directory_iterator end_it;

using namespace std;
using namespace cv;



// <editor-fold defaultstate="collapsed" desc="Parameter definitions">
/* Parameter definitions */

// Directory containing positive sample images

static string posSamplesDir = "/home/jose/Documentos/Entrenamiento_SVM/Pictures/
Positives(copia)/";

static string negSamplesDir = "/home/jose/Documentos/Entrenamiento_SVM/Pictures/
Negatives(copia)/";
// Set the file to write the features to
static string featuresFile = "/home/jose/Documentos/Entrenamiento_SVM/Features.dat";
// Set the file to write the prediction to
static string PredictionFile = "/home/jose/Documentos/Entrenamiento_SVM/
Predictions.dat";
// Set the file to write the SVM model to
static string svmModelFile = "/home/jose/Documentos/Entrenamiento_SVM/
svm_lightmodel.dat";
// Set the file to write the resulting detecting descriptor vector to
static string descriptorVectorFile = "/home/jose/Documentos/Entrenamiento_SVM/
descriptorvector.dat";

// HOG parameters for training that for some reason are not included in the HOG class
static const Size trainingPadding = Size(0, 0);
static const Size winStride = Size(8, 8);
// </editor-fold>

// <editor-fold defaultstate="collapsed" desc="Helper functions">
/* Helper functions */


 static string toLowerCase(const string& in) {
     string t;
     for (string::const_iterator i = in.begin(); i != in.end(); ++i) {
         t += tolower(*i);
     }
     return t;
 }

 static void storeCursor(void) {
     printf("\033[s");
 }

 static void resetCursor(void) {
     printf("\033[u");
 }
```

```
void resize_imagePositive(void) {

    cout<<"Buscando a partir de"<<directorioP.string().data()<<" "<<endl;

    if (! fs::is_directory("/home/jose/Documentos/Entrenamiento_SVM/Pictures"/
redimensionadoP)){
        fs::create_directory("/home/jose/Documentos/Entrenamiento_SVM/Pictures"/
redimensionadoP);

    }
    for( fs::directory_iterator it( directorioP ); it != end_it; it++ )
    {

        cout << "Itero en directorio" << endl;
        if( fs::is_regular_file( it->status() ) && fs::extension(it->path()) == ".png"
|| ".jpg" || ".jpeg")
        {
            std::cout << it->path().filename() << std::endl;


             image = cvLoadImage( it->path().string().data() );

             if ( image.empty() )
             {
                  cout << "Error al cargar la imagen" << endl;
             }

                    resize (image, image_red, size,0.5,0.5,INTER_LINEAR);
                    nombreOriginal= fs::unique_path("/home/jose/Documentos/
Entrenamiento_SVM/Pictures"/redimensionadoP/ (it->path().stem().string() +".png"));
                    cout << "Guardo fichero  " << nombreOriginal.string() << "
redimensionado " << endl;
                    cv::imwrite(nombreOriginal.string().data(),image_red);


    }
        image.release();
        image_red.release();
    }
    cout<<"LAS IMAGENES POSITIVAS HAN SIDO REDIMENSIONADAS"<<endl;
}


void resize_imageNegative(void) {

    cout<<"Buscando a partir de"<<directorioN.string().data()<<" "<<endl;

    if (! fs::is_directory("/home/jose/Documentos/Entrenamiento_SVM/Pictures"/
redimensionadoN)){
        fs::create_directory("/home/jose/Documentos/Entrenamiento_SVM/Pictures"/
redimensionadoN);

    }
    for( fs::directory_iterator it( directorioN ); it != end_it; it++ )
    {

        cout << "Itero en directorio" << endl;
        if( fs::is_regular_file( it->status() ) && fs::extension(it->path()) == ".png"
|| ".jpg" || ".jpeg")
        {
            std::cout << it->path().filename() << std::endl;


             image = cvLoadImage( it->path().string().data() );

             if ( image.empty() )
             {
                  cout << "Error al cargar la imagen" << endl;
```

```
            }

                    resize (image, image_red, size,0.5,0.5,INTER_LINEAR);
                    nombreOriginal= fs::unique_path("/home/jose/Documentos/
  Entrenamiento_SVM/Pictures"/redimensionadoN/ (it->path().stem().string() +".png"));
                    cout << "Guardo fichero  " << nombreOriginal.string() << "
  redimensionado " << endl;
                    cv::imwrite(nombreOriginal.string().data(),image_red);


      }
          image.release();
          image_red.release();
      }
      cout<<"LAS IMAGENES NEGATIVAS HAN SIDO REDIMENSIONADAS"<<endl;
  }


/**
 * For unixoid systems only: Lists all files in a given directory and returns a vector
of path+name in string format
 * @param dirName
 * @param fileNames found file names in specified directory
 * @param validExtensions containing the valid file extensions for collection in lower
case
 * @return
 */
static void getFilesInDirectory(const string& dirName, vector<string>& fileNames, const
vector<string>& validExtensions) {
    printf("Opening directory %s\n", dirName.c_str());
    struct dirent* ep;
    size_t extensionLocation;
    DIR* dp = opendir(dirName.c_str());
    if (dp != NULL) {
        while ((ep = readdir(dp))) {
            // Ignore (sub-)directories like . , .. , .svn, etc.
            if (ep->d_type & DT_DIR) {
                continue;
            }
            extensionLocation = string(ep->d_name).find_last_of("."); // Assume the last
point marks beginning of extension like file.ext
            // Check if extension is matching the wanted ones
            string tempExt = toLowerCase(string(ep->d_name).substr(extensionLocation +
1));
            if (find(validExtensions.begin(), validExtensions.end(), tempExt) !=
validExtensions.end()) {
                printf("Found matching data file '%s'\n", ep->d_name);
                fileNames.push_back((string) dirName + ep->d_name);
            } else {
                printf("Found file does not match required file type, skipping: '%s'\n",
ep->d_name);
            }
        }
        (void) closedir(dp);
    } else {
        printf("Error opening directory '%s'!\n", dirName.c_str());
    }
    return;
}


 static void calculateFeaturesFromInput(const string& imageFilename, vector<float>&
 featureVector, HOGDescriptor& hog) {
     /** for imread flags from openCV documentation,
      * @see http://docs.opencv.org/modules/highgui/doc/
 reading_and_writing_images_and_video.html?highlight=imread#Mat imread(const string&
 filename, int flags)
      * @note If you get a compile-time error complaining about following line (esp.
 imread),
      * you either do not have a current openCV version (>2.0)
      * or the linking order is incorrect, try g++ -o openCVHogTrainer main.cpp `pkg-
```

```
  config --cflags --libs opencv`
       */



    std::cout << "Calculating features from " << imageFilename << std::endl;

    Mat imageData = imread(imageFilename, 0);
    if (imageData.empty()) {
        featureVector.clear();
        printf("Error: HOG image '%s' is empty, features calculation skipped!\n",
imageFilename.c_str());
        return;
    }
    // Check for mismatching dimensions
    if (imageData.cols != hog.winSize.width || imageData.rows != hog.winSize.height) {
        featureVector.clear();
        printf("Error: Image '%s' dimensions (%u x %u) do not match HOG window size (%u
x %u)!\n", imageFilename.c_str(), imageData.cols, imageData.rows, hog.winSize.width,
hog.winSize.height);
        return;
    }
    vector<Point> locations;
    hog.compute(imageData, featureVector, winStride, trainingPadding, locations);

    imageData.release(); // Release the image again after features are extracted
}

/**
 * Shows the detections in the image
 * @param found vector containing valid detection rectangles
 * @param imageData the image in which the detections are drawn
 */
static void showDetections(const vector<Rect>& found, Mat& imageData) {

    cout << "Detectados: " << found.size() << endl;
    vector<Rect> found_filtered;
    size_t i, j;
    for (i = 0; i < found.size(); ++i) {
        Rect r = found[i];
        for (j = 0; j < found.size(); ++j)
            if (j != i && (r & found[j]) == r)
                break;
         if (j == found.size())
            found_filtered.push_back(r);
    }

    for (i = 0; i < found_filtered.size(); i++) {
        Rect r = found_filtered[i];
        rectangle(imageData, r.tl(), r.br(), Scalar(64, 255, 64), 3);

        cout <<"Deteccion numero "<<i+1<< r.tl()<<"    " <<"altura=" << r.height<< "   "
<< "Anchura= " <<r.width<< "\n";

        ofstream fs("/home/jose/Documentos/Entrenamiento_SVM/imagenes_etiquetadas/
coordenadas_imagen_numero_1.txt"); // Fichero donde guardar coordenadas alturas y
anchuras.

        fs<<"Deteccion numero "<<i+1<< r.tl()<<"    " <<"altura=" << r.height<< "   " <<
"Anchura= " <<r.width<< "\n"; //para ver el valor de las coordenadas y

// compararlas con las del etiquetador

    }
}

/**
 * Test detection with custom HOG description vector
 * @param hog
```

```
 * @param imageData
 */
static void detectTest(const HOGDescriptor& hog, Mat& imageData) {

    vector<Rect> found;
    int groupThreshold = 2; // 2
    Size padding(Size(32, 32));
    Size winStride(Size(8, 8));
    double hitThreshold = -1; // tolerance -1
    hog.detectMultiScale(imageData, found, hitThreshold, winStride, padding, 1.05,
groupThreshold);
    showDetections(found, imageData);
}
// </editor-fold>



static void saveDescriptorVectorToFile(vector<float>& descriptorVector, vector<unsigned
int>& _vectorIndices, string fileName) {
    printf("Saving descriptor vector to file '%s'\n", fileName.c_str());
    string separator = " "; // Use blank as default separator between single features
    fstream File;
    float percent;
    File.open(fileName.c_str(), ios::out);
    if (File.good() && File.is_open()) {
        printf("Saving %lu descriptor vector features:\t", descriptorVector.size());
        storeCursor();
        for (int feature = 0; feature < descriptorVector.size(); ++feature) {
            if ((feature % 10 == 0) || (feature == (descriptorVector.size()-1)) ) {
                percent = ((1 + feature) * 100 / descriptorVector.size());
                printf("%4u (%3.0f%%)", feature, percent);
                fflush(stdout);
                resetCursor();
            }
            File << descriptorVector.at(feature) << separator;
        }
        printf("\n");
        File << endl;
        File.flush();
        File.close();
    }
}



 /**
  * Main program entry point
  * @param argc
  * @param argv
  * @return EXIT_SUCCESS (0) or EXIT_FAILURE (1)
  */
 int main(int argc, char** argv) {


 //Para redimensionar las imagenes que no sean 64x128
    // resize_imagePositive();
   //  resize_imageNegative();

     // <editor-fold defaultstate="collapsed" desc="Init">

     HOGDescriptor hog; // Use standard parameters here
     hog.winSize = Size(64, 128);



   // Get the files to train from somewhere
  static vector<string> positiveTrainingImages;
  static vector<string> negativeTrainingImages;
  static vector<string> validExtensions;
  validExtensions.push_back("png");
  validExtensions.push_back("jpg");
```

```
    getFilesInDirectory(posSamplesDir, positiveTrainingImages, validExtensions);
    getFilesInDirectory(negSamplesDir, negativeTrainingImages, validExtensions);
    /// Retrieve the descriptor vectors from the samples
    unsigned long overallSamples = positiveTrainingImages.size() +
negativeTrainingImages.size();

    // Make sure there are actually samples to train
    if (overallSamples == 0) {
        printf("No training sample files found, nothing to do!\n");
        return EXIT_SUCCESS;
    }

    //Check nr of features for dimensions
    vector<float> testVector;
    calculateFeaturesFromInput(positiveTrainingImages[0].data(), testVector, hog);
    size_t featuresNr=testVector.size();
    std::cout << "el numero de features de la imagen " <<
positiveTrainingImages[0].data() << " es " << featuresNr << std::endl;
    std::cout << "el numero de imagenes es  " << overallSamples << std::endl;

    float labels[overallSamples];




    std::cout << "intento crear un array de   " << overallSamples  << " por " <<
featuresNr << std::endl;


//cuando el numero de imagenes es muy grande hay que crear la tabla como static y con el
numero concreto en tiempo de compilacion
    static float trainingData[31141] [3780];
    //float trainingData[overallSamples] [featuresNr];

    int index=0;

    std::cout << "el numero de imagenes es  " << overallSamples << std::endl;



    /// @WARNING: This is really important, some libraries (e.g. ROS) seems to set the
system locale which takes decimal commata instead of points which causes the file input
parsing to fail
    setlocale(LC_ALL, "C"); // Do not use the system locale
    setlocale(LC_NUMERIC,"C");
    setlocale(LC_ALL, "POSIX");

    printf("Reading files, generating HOG features and save them to file '%s':\n",
featuresFile.c_str());
    float percent;
    /**
     * Save the calculated descriptor vectors to a file in a format that can be used by
SVMlight for training
     * @NOTE: If you split these steps into separate steps:
     * 1. calculating features into memory (e.g. into a cv::Mat or vector< vector<float>
>),
     * 2. saving features to file / directly inject from memory to machine learning
algorithm,
     * the program may consume a considerable amount of main memory
```

```
    fstream File;
    File.open(featuresFile.c_str(), ios::out);

    fs::path suffix (".dat");

    fstream singleFile;

    if (File.good() && File.is_open()) {
        // Remove following line for libsvm which does not support comments
        // File << "# Use this file to train, e.g. SVMlight by issuing $ svm_learn -i 1
-a weights.txt " << featuresFile.c_str() << endl;
        // Iterate over sample images
        for (unsigned long currentFile = 0; currentFile < overallSamples; ++currentFile)
{
            storeCursor();
            vector<float> featureVector;
            // Get positive or negative sample image file path
            const string currentImageFile = (currentFile < positiveTrainingImages.size()
? positiveTrainingImages.at(currentFile) : negativeTrainingImages.at(currentFile -
positiveTrainingImages.size()));
            // Output progress
            if ( (currentFile+1) % 10 == 0 || (currentFile+1) == overallSamples ) {
                percent = ((currentFile+1) * 100 / overallSamples);
                printf("%5lu (%3.0f%%):\tFile '%s'", (currentFile+1), percent,
currentImageFile.c_str());
                fflush(stdout);
                resetCursor();
            }
            // Calculate feature vector from current image file
            calculateFeaturesFromInput(currentImageFile, featureVector, hog);
            if (!featureVector.empty()) {
                /* Put positive or negative sample class to file,
                 * true=positive, false=negative,
                 * and convert positive class to +1 and negative class to -1 for
SVMlight
                 */
                fs::path fichero(currentImageFile);
                singleFile.open((fichero.string()+ suffix.string()).data(), ios::out);

                 if (singleFile.good() && singleFile.is_open()) {



         //       std::cout << "Guardando HOG individual en " << (fichero.string()+
suffix.string()).data() << std::endl;
                 }
                 else
                 {
                     std::cout << "=========== ERROR guardando HOG individual en  " <<
(fichero.string()+ suffix.string()).data() << std::endl;
                 }



                File << ((currentFile < positiveTrainingImages.size()) ? "+1" : "-1");
                singleFile << ((currentFile < positiveTrainingImages.size()) ? "+1" :
"-1");
                // Save feature vector components
                for (unsigned int feature = 0; feature < featureVector.size(); +
+feature) {
                    File << " " << (feature + 1) << ":" << featureVector.at(feature);
                    singleFile << " " << (feature + 1) << ":" <<
featureVector.at(feature);
                }
                File << endl;
                singleFile << endl;


                labels[index] =((currentFile < positiveTrainingImages.size()) ? 1.0 :
```

```
-1.0);

                for (size_t i=0 ; i!=featuresNr; i++)
                {
                    trainingData[index][i]=featureVector.at(i);
                }
                index++;


            }
            singleFile.flush();
            singleFile.close();
        }
        printf("\n");
        File.flush();
        File.close();

    } else {
        printf("Error opening file '%s'!\n", featuresFile.c_str());
        return EXIT_FAILURE;
    }


    std::cout<< " \n TRAINING... \n"<< std::endl;

    Mat labelsMat(overallSamples,1, CV_32FC1, labels);
    Mat trainingDataMat(overallSamples, featuresNr, CV_32FC1, trainingData);

    std::cout<< " \n mat creados... \n"<< std::endl;


    // Set up SVM's parameters
       CvSVMParams params;
       params.svm_type    = CvSVM::C_SVC;
       params.kernel_type = CvSVM::LINEAR; //LINEAR;
       params.term_crit   = cvTermCriteria(CV_TERMCRIT_ITER, 100, 1e-6); //100, 1e -6



        cout << "Llamo al entrenamiento" << endl;
        // Train the SVM
        CvSVM SVM;
        SVM.train(trainingDataMat, labelsMat, Mat(), Mat(), params);

        std::cout<< " \n TRAINED!!! \n"<< std::endl;

       // SVM.save(svmModelFile.data()) ;//YO

        SVM.save(descriptorVectorFile.data());



         //Aqui pruebo a predecir con las muestras
         fstream File2;
         File2.open(PredictionFile.c_str(), ios::out);
         if (File2.good() && File2.is_open()) {
      for (size_t i=0; i < overallSamples;  i++)
         {
         std::cout << i << " es " << labels[i] << " y predigo " <<
SVM.predict(trainingDataMat.row(i)) << std::endl;
         File2 << i << " es " << labels[i] << " y predigo " <<
SVM.predict(trainingDataMat.row(i)) << std::endl;
      }
               File2<<endl;
               File2.flush();
               File2.close();
```

```
        }else {
            printf("Error opening file '%s'!\n", PredictionFile.c_str());
            return EXIT_FAILURE;
        }

//*



        hog.setSVMDetector(descriptorVector);


        std::cout << "ahora pruebo a detectar en una imagen completa" << endl;

        Mat aDetectar;
        aDetectar = cv::imread(imagenADetectar.string());
        //namedWindow("original");
        namedWindow("detecciones");
        //imshow ("original", aDetectar);
        //ahora pruebo a detectar en una imagen completa
        detectTest( hog, aDetectar);
        imshow ("detecciones", aDetectar);
        cv::waitKey(0);
//*/


    return EXIT_SUCCESS;
}
```

Bibliografía

[1] Dalal, Navneet; Triggs, Bill. "*Histograms of oriented gradients for human detection*". En *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. IEEE, 2005. p. 886-893.

[2] Morán Cruz, Natalia. "*Desarrollo de un sistema avanzado de asistencia a la conducción en tiempo real para la detección de peatones en entornos urbanos complejos*"

[3] Intxaurbe Txarterina, Jon. "*Detección de personas*".

[4] DGT, "*Anuario Estadístico de Accidentes 2014*"

[5] Carsten, O. M. ., & Nilsson, L. (2001). "*Safety Assessment of Driver Assistance Systems. European Journal of Transport and Infrastructure*" Research, 1(3), 225–243. Retrieved from http://eprints.whiterose.ac.uk/2007/ (Última consulta 23/09/2016)

[6] Zelinsky, a. (2009). Learning OpenCV---Computer Vision with the OpenCV Library (Bradski, G.R. et al.; 2008)[On the Shelf]. IEEE Robotics {&} Automation Magazine (Vol. 16tane). http://doi.org/10.1109/MRA.2009.933612 (Última consulta 23/09/2016)

[7] CEA. "*Seguridad activa y pasiva del vehículo*" <http://www.cea-online.es/reportajes/seguridad.asp > (Última consulta 23/09/2016)

[8] VPE Seguridad Vial. "*Implantación de los Sistemas de Asistencia a la Conducción Avanzados (ADAS) en el mercado español*". http://www.vpeseguridadvial.com/implantacion-de-los-sistemas-de-asistencia-al-conductor-en-el-mercado-espanol/ (Última consulta 23/09/2016)

[9] Gietelink, O., Ploeg, J., De Schutter, B., & Verhaegen, M. (2006). "*Development of advanced driver assistance systems with vehicle hardware-in-the-loop simulations*". Vehicle System Dynamics, 44(7), 569–590.

[10] CIRCULA SEGURO. *"¿Qué son los sistemas de visión nocturna?"* <http://www.circulaseguro.com/que-son-los-sistemas-de-vision-nocturna/> (Última consulta 23/09/2016)

[11] Megane, Dani. "*Sistema de Aparcamiento Asistido*" <http://www.aficionadosalamecanica.com/sistema-aparcamiento-asistido.htm> (Última consulta 23/09/2016)

[12]TOYOTA, "*Asistencia al aparcamiento*" <https://www.toyota.es/world-of-toyota/safety-technology/parking-aids.json> (Última consulta 23/09/2016)

[13]CIRCULA SEGURO. "*¿Qué son las luces adaptativas y automáticas?*" < http://www.circulaseguro.com/que-son-las-luces-adaptativas-y-automaticas/> (Última consulta 23/09/2016)

[14] Cisneros, Óscar "*Los sistemas de Adaptación Inteligente de la Velocidad*" <http://www.centro-zaragoza.com:8080/web/sala_prensa/revista_tecnica/hemeroteca/articulos/R38_A7.pdf> (Última consulta 23/09/2016)

[15] FEUVERT " El sistema de adaptación inteligente de velocidad ISA divide a Europa" <http://www.feuvertenmarcha.org/el-sistema-de-adaptacion-inteligente-de-velocidad-isa-divide-a-europa/> (Última consulta 23/09/2016)

[16] BOSCH "Seguridad Vial y Medio ambiente". < <http://www.anfac.com/openPublicPdf.action?idDoc=7883> (Última consulta 23/09/2016)

[17] Pérez Cazorla, José Antonio "*Como funciona el sistema de detección de Ángulo muerto*" <http://www.buscadordetalleres.com/blog/como-funciona-el-sistema-de-deteccion-del-angulo-muerto/> (Última consulta 23/09/2016)

[18] E-AUTO http://e-auto.com.mx/manual_detalle.php?manual_id=249 (Última consulta 23/09/2016)

[19]HITECH. http://hitechautomechanical.com.au/Front-Rear-Brake-Replacement-ABS-Service (Última consulta 23/09/2016)

[20] FITSA "*Tecnologías vehiculares para la mejora de la protección de peatones y ciclistas*" http://creandoconciencia.org.ar/enciclopedia/salud/PEATONES.pdf (Última consulta 23/09/2016)

[21] TECMOVIA "*Los sistemas de protección para peatones avanzan con paso firme*" http://www.diariomotor.com/tecmovia/2014/04/16/los-sistemas-de-proteccion-para-peatones-avanzan-con-paso-firme-laboratorio-tecmovia/ (Última consulta 23/09/2016)

[22]Flores,Javier < http://www.muyinteresante.es/innovacion/articulo/google-crea-su-propio-coche-autonomo-881401361097> (Última consulta 23/09/2016)

[23]Markoff John, "*Google Cars drive themselves*" http://www.nytimes.com/2010/10/10/science/10google.html?_r=1 (Última consulta 23/09/2016)

[24]20 MINUTOS <http://www.20minutos.es/noticia/2134853/0/coche-inteligente/universidad-carlos-tercerno/peatones-noche/ >

[25]Intelligent Systems Labortory http://portal.uc3m.es/portal/page/portal/dpto_ing_sistemas_automatica/investigacion/IntelligentSystemsLab/research/augv (Última consulta 23/09/2016).

[26] Musleh, B., de la Escalera, A., & Armingol, J. M. (2012). "*Detección de obstáculos y espacios transitables en entornos urbanos para sistemas de ayuda a la conducción basados en algoritmos de visión estéreo implementados en GPU*". RIAI - Revista Iberoamericana de Automática E Informática Industrial, 9(4), 462–473.

[27] Rodríguez Garavito, C. H., Ponz, A., García, F., Martín, D., Escalera, A. de la, & Armingol, J. M. (2014). "*Automatic Laser And Camera Extrinsic Calibration for Data Fusion Using Road Plane*". Information Fusion (FUSION), 2014.

[28] Viola, P., & Jones, M. (2004). "*Robust Real-Time Face Detection*". International Journal of Computer Vision, 57, 137–154. http://doi.org/10.1023/B:VISI.0000013087.49260.fb

[29] Alguacil Gómez, Abel "*Aplicaciones del operador SIFT al reconocimiento de objetos*"

[30] Bay, H., Tuytelaars, T., & Van Gool, L. (2016, January). "*Speeded up robust features*". Lecture Notes in Computer Science. http://doi.org/10.1007/11744023_32

[31] Jaramillo, M. A., Fernández, J. Á., & Martínez De Salazar, E. (2000). "*Implementación del Detector de Bordes de Canny sobre Redes Neuronales Celulares*". Proc. SAAEI , 10, 285–288. http://eii.unex.es/profesores/jalvarof/pubs.htm\nhttp://eii.unex.es/profesores/jalvarof/pdf/canny00.pdf (Última consulta 23/09/2016)

[32]Malik, J., Belongie, S., Leung, T., & Shi, J. (2001). "*Contour and texture analysis for image segmentation*". International Journal of Computer Vision, 43(1), 7–27. http://doi.org/10.1023/A:1011174803800 (Última consulta 23/09/2016)

[33] Rssohu, H., Zlwk, H., Chiu, Y., Wang, F., Chou, Y., Horng, T., Guardino, C. (2014). "*Hand gesture recognition using combined features of location, angle and velocity*". Pattern Recognition, 34(4), 1. http://doi.org/10.1016/S0031-3203(00)00096-0 (Última consulta 23/09/2016)

[34] S. Kotsiantis, "*Supervisado Aprendizaje Automático: Una Revisión de la Clasificación de las técnicas de Informática*". Diario 31 (2007).

[35] Viola, P. a, & Jones, M. J. (2001). "*Fast and Robust Classification using Asymmetric AdaBoost and a Detector Cascade*". Advances in Neural Information Processing System, (December), 1311–1318. Retrieved from https://papers.nips.cc/paper/2091-fast-and-robust-classification-using-asymmetric-adaboost-and-a-detector-cascade.pdf (Última consulta 23/09/2016)

[36] B. W. SILVERMAN AND M. C. JONES, "*E. fix and jl hodges (1951): an important contributionto nonparametric discriminant analysis and density estimation: commentaryon fix and hodges (1951)*," International Statistical Review/Revue Internationale de Statistique, pp. 233–238, 1989.

[37] Vapnik, V. (1995). "*Support vector machine*". Machine Learning, 20(3), 273–297. Retrieved from http://mlab.cb.k.u-tokyo.ac.jp/~moris/lecture/cb-mining/4-svm.pdf\nfile:///Users/atb/Documents/Library.papers3/Files/Machine learning 1995 Vapnik.pdf\npapers3://publication/uuid/5AC02E6D-E5D6-40BC-9188-73C50CB8FCE1

[38] Elliott, R. J., & Aggoun, L. (1996). "*Estimation for hidden Markov random fields*". Journal of Statistical Planning and Inference, 50(3), 343–351.

[39] MacQueen, J. B. (1967). "*Kmeans Some Methods for classification and Analysis of Multivariate Observations*". 5th Berkeley Symposium on Mathematical Statistics and Probability 1967, 1(233), 281–297. http://doi.org/citeulike-article-id:6083430 (Última consulta 23/09/2016)

[40] Kohonen, T. (2001). "*Self-Organizing Maps*". Springer Series in Information Sciences, 30. http://doi.org/10.1007/978-3-642-56927-2 (Última consulta 23/09/2016)

[41] Valveny, Ernest. "*Detector basado en HOG/SVM*" https://es.coursera.org/learn/deteccion-objetos/lecture/WSJ9t/l4-2-hog-calculo-del-gradiente (Última consulta 23/09/2016)

[42]Valveny, Ernest. "*Cálculo de Histogramas*" https://es.coursera.org/learn/deteccion-objetos/lecture/GBJYS/l4-3-hog-calculo-de-los-histogramas (Última consulta 23/09/2016)

[43] Valveny, Ernest. "*Cálculo del descriptor*"https://es.coursera.org/learn/deteccion-objetos/lecture/uAEKB/l4-4-hog-calculo-del-descriptor (Última consulta 23/09/2016)

[44] Alba Castro, José Luis. "*Máquinas de Vectores Soporte (SVM)*"

[45] Carmona, Enrique http://www.ia.uned.es/~ejcarmona/publicaciones/[2013-Carmona]%20SVM.pdf (Última consulta 23/09/2016)

[46]EFFIDENCE, http://effistore.effidence.com/capteurs-mesures/sensors/scanners-lasers/sick-ld-mrs.html?___store=english&___from_store=default; DIRECT INDUSTRY http://www.directindustry.com/prod/point-grey/product-100807-920151.html (Última consulta 23/09/2016)

[47]Tsochantaridis, I., Hofmann, T., Joachims, T., & Altun, Y. (2004). "*Support vector machine learning for interdependent and structured output spaces*". Internation Conference on Machine Learning, 104. Retrieved from http://portal.acm.org/citation.cfm?doid=1015330.1015341 (Última consulta 23/09/2016)

[48] UBUNTU <http://tecnologia.starmedia.com/tutoriales/requisitos-minimos-para-instalar-ubuntu-en-pc.html > (Última consulta 23/09/2016)

[49]QTCREATORhttp://bibing.us.es/proyectos/abreproy/90079/fichero/tfg-AnaMariaEscorzaAguilar.pdf (Última consulta 23/09/2016)